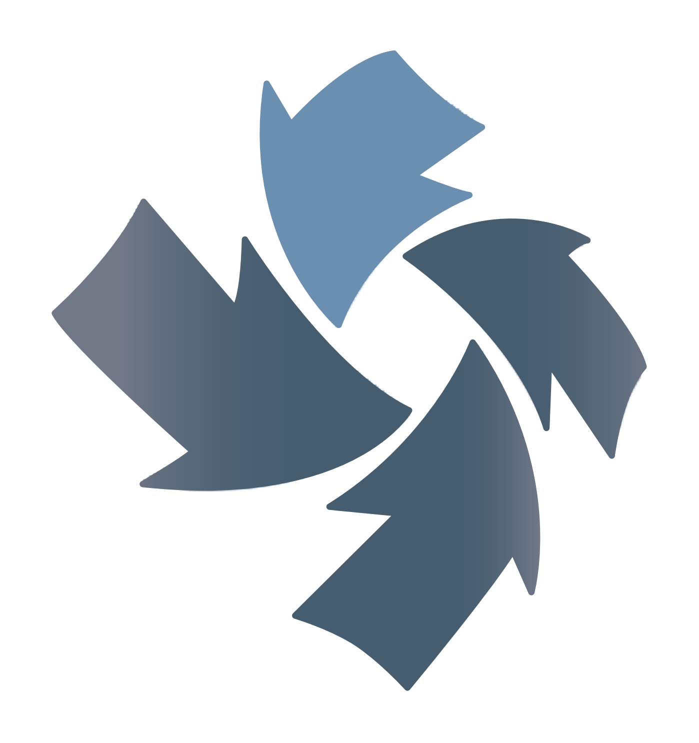




OpenDataHub HSR – Teilen, Beziehen und Konvertieren von Daten

Bachelorarbeit im Frühjahrssemester 2015



Autoren
Remo Liebi
Christoph Hüsler
Fabio Scala

Betreuer
Prof. Stefan Keller
Experte
Claude Eisenhut
Gegenleser
Prof. Dr. Markus Stolze

Rapperswil, 1. Juli 2015

Erstellungsdatum	11. Februar 2015 13:03
Änderungsdatum	1. July 2015 22:10
Druckdatum	1. Juli 2015 22:19
Bezugsadresse	`https://hsr-ba-fs15-dat.github.io/ba-doc/main.pdf`
Version	459
Git Hash	0cf4e61d5cfbee2ea4afa3ddd9b6abb601df8686
Autoren	Christoph Hüsler chuesler@hsr.ch Fabio Scala fabio.scala@gmail.com Remo Liebi remo@liebi.net
Lizenz	(cc) 2015

Abstract

Im digitalen Zeitalter spielen Daten jeglicher Art eine zentrale Rolle für die reibungslose Zusammenarbeit zwischen diversen Organisationen und deren Applikationen. Diese Daten kommen in einer Vielzahl an Formaten und unterschiedlichen Schemata daher, welche erst durch einen komplexen und oftmals individuellen Transformationsprozess zu eigenen Zwecken wiederverwendet werden können. Ziel dieser Arbeit ist die Abstrahierung von Dateiformat und Schema-Transformation, um einen Datenaustausch zwischen diversen Parteien mit Hilfe einer Webapplikation zu vereinfachen – OpenDataHub.ch.

Nach Evaluation von vorgegebenen (dat-data.com) sowie weiteren Datenaustausch-Plattformen wurde ein modulares und erweiterbares Konzept zur Konversion diverser Dateiformate sowie die Transformation mittels einer eigenen Domain Specific Language (DSL) mit dem Namen "OpenDataHub Query Language" (ODHQL) entworfen und umgesetzt. Die ODHQL wurde aufgrund des bereits vorhandenen SQL Know-hows vieler Entwickler als Subset dessen umgesetzt. Die funktionalen Anforderungen an die Plattform sowie die ODHQL wurde durch die Umsetzung zweier Anwendungsfälle getrieben: Die Integration von Postadressen und Verkehrshindernisse für die Traffic Obstruction Database (TROBDB).

Das Resultat ist eine moderne, mit Python 2.7, Django Framework und AngularJS umgesetzte HTML5-Webapplikation, mit der Daten diverser Formate öffentlich geteilt und durch Experten mit ODHQL-Kenntnissen transformiert werden können. Diese lassen sich dann wiederum zur Weiterverwendung in einem beliebigen Dateiformat beziehen.

Die Erkenntnisse und das Resultat dieser Arbeit bestätigen die Machbarkeit einer solchen Konversions- und Transformationsplattform und können als Grundlage für weitere Entwicklungen verwendet werden.

Webseite: beta.opendatahub.ch

Management Summary

Ausgangslage

Den Schweizer Behörden sowie Drittparteien steht die Wahl der eingesetzten Software-Systeme frei. Diese produzieren wiederum verschiedene Dateiformate und Schemata zum Austausch der Daten. Durch den Einsatz einer Vielzahl an Systemen wird der Datenaustausch massiv erschwert. Es wurde versucht, dieses Problem mit dem Daten-Austauschformat Interlis zu lösen. Aufgrund mangelnder Unterstützung seitens der Software-Hersteller muss jedoch nach wie vor erst eine Konversion in dieses Format vor dem Austausch durch die beteiligten Stellen selbst vorgenommen werden.

Dies bringt einen beachtlichen technischen und betrieblichen Aufwand mit sich.

Ziele

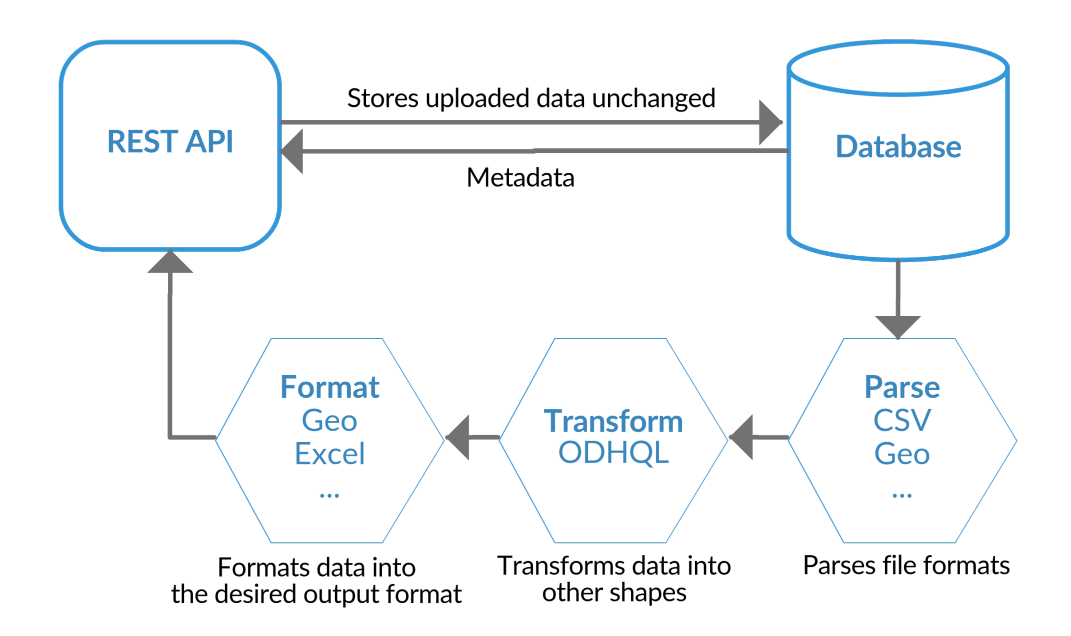
Das Ziel dieser Arbeit ist es, einen Prototypen für eine Plattform zu evaluieren und implementieren, welche

- diverse Dateiformate lesen und schreiben kann, so dass ein Benutzer sich um die Daten kümmern kann, statt sich mit Formaten befassen zu müssen;
- fortgeschrittenen Benutzern erlaubt, neue Sichten oder Transformationen für bereitgestellte Daten zu erstellen, welche andere Benutzer dann weiter verwenden können;
- Mit Geo-Daten umgehen kann;
- Benutzern ermöglicht, Daten öffentlich anzubieten.

Für das reine Anbieten von Daten existieren bereits gute Lösungen (siehe **??**). Aus diesem Grund fokussiert sich diese Arbeit auf die Transformation von Daten sowie Formatunterstützung.

Ergebnisse

Im Rahmen der Arbeit wurde mit OpenDataHub (ODH) eine moderne Webapplikation geschaffen.



Übersicht OpenDataHub Architektur

Formate

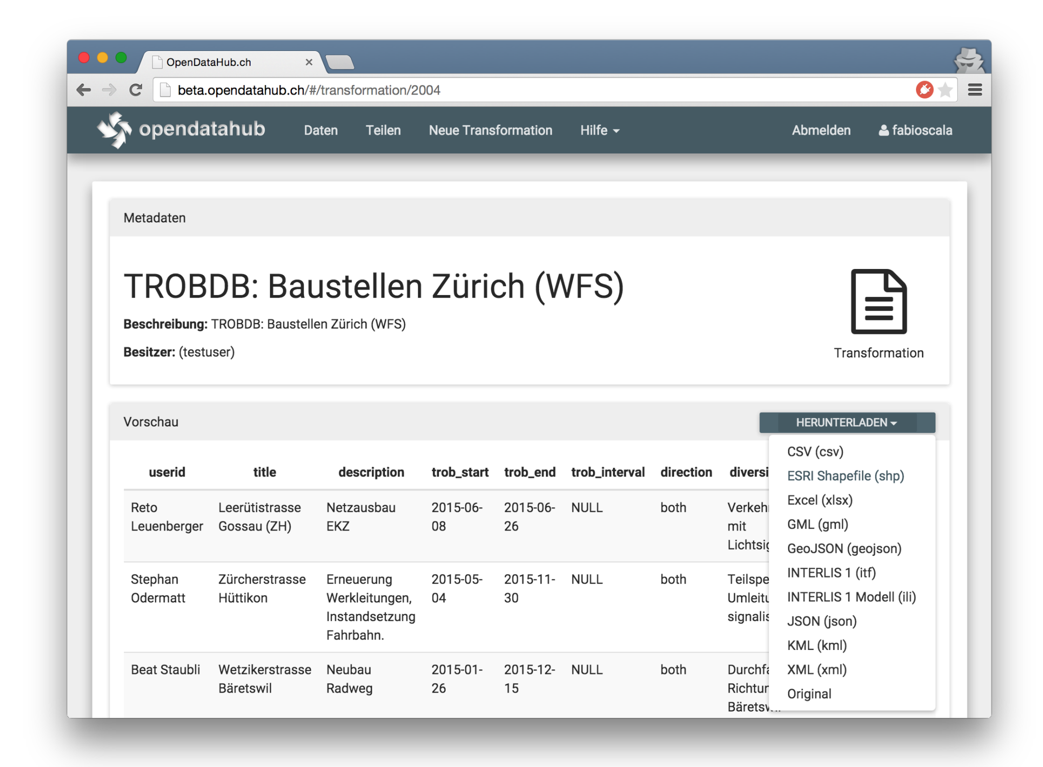
Benutzer können Daten mit anderen Benutzern teilen, ohne sich um Format-Details kümmern zu müssen. Unterstützt werden Formate wie CSV, Excel und JSON, aber auch Geo-Formate wie GeoJSON, GML, Interlis 1 sowie Daten aus Web Feature Service (WFS)-Endpunkten.

Des Weiteren besteht die Möglichkeit, aus einer Analyse der bereit gestellten Daten ein Interlis 1-Modell zu erstellen.

Transformationen

Mit Hilfe eines Assistenten ist es Benutzern möglich die Grundlage einer Transformation auf einfache Art und Weise zu erstellen.

Für Experten steht mit OpenDataHub Query Language (ODHQL) eine an SQL angelehnte Abfrage-Sprache bereit, welche auch komplexere Abfragen ermöglicht.



Resultat einer Transformation

Die Resultate von Transformationen können wie alle anderen Daten auch in beliebigen Formaten bezogen werden.

Erweiterung

Neue Formate sowie Funktionen für die Abfrage-Sprache können mit relativ wenig Aufwand hinzugefügt werden.

Ausblick

Für die Weiterentwicklung von OpenDataHub bestehen diverse Möglichkeiten. Hier eine Auswahl:

- Wechsel auf neue Versionen der zur Format-Unterstützung verwendeten Bibliotheken.
- Erweiterung der Web-Applikation, evtl. Integration mit bereits vorhandener Software.
- Ausbau des Transformations-Assistenten um weitere Features.
- Unterstützung für weitere Formate.



OpenDataHub HSR – Teilen, Beziehen und Konvertieren von Daten

Bachelorarbeit Frühlingssemester 2015, Studiengang Informatik

Aufgabenstellung

Der Austausch von strukturierten Daten - d.h. der Datenaustausch – zwischen unabhängigen Organisationen ist nicht einfach. In einer idealen Welt einigt man sich auf einen gemeinsamen Datenaustausch-Mechanismus, beschreibt Datenmodelle für verschiedene Anwendungsbereiche und jedes beteiligte System ist dafür besorgt, dass die Daten gemäss diesem Modell in einem einheitlichen Format importiert, bearbeitet und wieder exportiert werden können.

Die Realität scheint sich immer weiter von dieser Vision zu entfernen. Die marktbeherrschenden Softwarehersteller überlassen das Problem den Nutzern. Die Datenlieferanten und Systembetreiber sind überfordert und einigen sich auf spezifische oder proprietäre Lösungen - oder sie begnügen sich mit 1:1-Konvertern (vgl. OGR und geoconverter.hsr.ch etc.) oder analogen Lösungen (Papier).

Ein Lösungsansatz könnte ein zentrales Portal (= "OpenDataHub") sein, über dieses sich Daten offen teilen, beziehen und konvertieren lassen wobei sich die Beteiligten nicht um die Formatkonversion und Schematransformation kümmern müssen: Die Datenlieferanten liefern ihre Daten in ihrem Format und Schema und die Nutzer beziehen es ihrerseits, wie sie wollen - vorausgesetzt, die entsprechende Formatkonversion und Schematransformation existiert. Darum kümmern sich Spezialisten.

Typische Rollen dieser Architektur sind:
- Datenlieferanten
- Nutzer
- Spezialisten für Schematransformationen

Typische Komponenten dieser Architektur sind:
- Lese- und Schreib-Komponenten
- Komponenten für typische Konversionsaufgaben
- Fertige "Transformationen" (vgl. Use Cases)

Use Cases

1. Postadressen (CSV)
2. Verkehrshindernisse-Datenbank
3. Gebäudeadressen (MOPublic)

Deliverables

- Überblick und Evaluation ähnlicher Projekte, insbesondere: „Project dat", CKAN (geopol.ch).
- Erwähnte Use Cases inkl. Beschreibung, (Demo-)Daten und Schema-Transformationen etc.
- Überlegungen und Evaluation zum Schema-Mapping: Sprache und Tools (Pentaho Kettle, ogrtools, FME, SQL, Eigene?)
- Die Implementation, die u.a. umfasst:
    - Listen-Anzeige bzw. einfaches Suchen/Filter von Daten
    - Publizieren von Daten
    - Storage (File oder Database, relational/NoSQL)
    - Beziehen von Daten



- Lesen/Schreiben ausgewählter Formate, insbesondere CSV, Shapefile, GeoPackage, GML, Interlis etc. (z.B. mittels OGR bzw. ogrtools)
- Schema-Mapping-Software

- Webseite mit (Cloud/Web-)Services mit Demonstration, mind. enthaltend die erwähnten Use Cases
- Angebot von vordefinierten (parametrisierbaren) Schema-Transformationen.

Vorgaben/Rahmenbedingungen

- Die Software soll Open Source und auf gängigen Browsern und Linux-Servern lauffähig sein.
- Die SW-Engineering-Methode und Meilensteine werden mit dem Betreuer vereinbart.
- Sourcecode und Software-Dokumentation sind Englisch (inkl. Installation, keine Benutzerdokumentation, höchstens eine Online-Kurzhilfe).
- Das Software-Frontend ist Deutsch (optional auch Englisch).
- Die Projekt-Dokumentation und -Präsentation sind auf Deutsch.
- Die Nutzungsrechte an der Arbeit bleiben bei den Autoren und gehen auch an die HSR/Betreuer über. Die Softwarelizenz ist „MIT“.
- (Ein Video - gemäss den Vorgaben des Studiengangs – ist optional).
- Ansonsten gelten die Rahmenbedingungen, Vorgaben und Termine des Studiengangs Informatik bzw. der HSR.

Inhalt der Dokumentation

- Die Projektdokumentation (Prosa) und die Benutzerschnittstelle sind in Deutsch. Der Code, die Kommentare und die Versionsverwaltung sind in Englisch.
- Die fertige Arbeit muss folgende Inhalte haben:
  1. Abstract, Management Summary, Aufgabenstellung
  2. Technischer Bericht
  3. Projektdokumentation
  4. Anhänge (Literaturverzeichnis, Glossar, CD-Inhalt)
- Die Abgabe ist so zu gliedern, dass die obigen Inhalte klar erkenntlich und auffindbar sind.
- Zitate sind zu kennzeichnen, die Quelle ist anzugeben.
- Verwendete Dokumente und Literatur sind in einem Literaturverzeichnis aufzuführen.
- Dokumentation des Projektverlaufes, Planung etc.
- Weitere Dokumente (z.B. Kurzbeschreibung, Poster) gemäss www.hsr.ch und Absprache mit dem Betreuer.

Form der Dokumentation

- Bericht (Struktur gemäss Beschreibung) gebunden (2 Exemplare) und in Ordner (1 Exemplar „kopierfähig“ in losen, gelochten Blättern).
- Alle Dokumente und Quellen der erstellten Software auf sauber angeschriebenen CD (3 Ex.).

Bewertungsschema

Es gelten die üblichen Regelungen zum Ablauf und zur Bewertung der Arbeit (6 Aspekte) des Studiengangs Informatik der HSR jedoch mit besonderem Gewicht auf moderne Softwareentwicklung (Tests, Continuous Integration, einfach installierbar, funktionsfähig).



Beteiligte

Diplomanden

Fabio Scala

Christoph Hüsler

Remo Liebi

Industriepartner

- (Open Community / Open Source).

Betreuung HSR

Verantwortlicher Dozent: Prof. Stefan Keller (sfkeller@hsr.ch), Geometa Lab am IFS der HSR

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

main.lot

Programmcodeverzeichnis

main.minted

Liste der Entscheidungen

main.decisions

Teil I

Technischer Bericht

1 Einführung

1.1 Vision

> Im Rahmen der Reform der Amtlichen Vermessung (RAV) war die Methodenfreiheit ein zentrales Stichwort. Man meinte damit insbesondere auch die Freiheit, ein für die Aufgabe geeignetes Programm-System frei wählen zu können, ohne deswegen Probleme mit dem Datenaustausch mit anderen Systemen zu haben und sagte darum: In der (öffentlichen) GeoSzene-Schweiz einigt man sich auf den Datenaustausch-Mechanismus Interlis. (aus [**rahmenvorst-interlisopen**])

Für den Datenaustausch kümmern sich die beteiligten Systeme dann nur noch um die Konvertierung von bzw. nach Interlis. Optimalerweise wird Interlis von den beteiligten Systemen bereits unterstützt.

Realität In der Realität interessieren sich die Anbieter von Geo-Software jedoch nicht für Interlis, welches praktisch ausschliesslich in der Schweiz verwendet wird. Als Folge davon muss die Konvertierung von und zu Interlis von den jeweils beteiligten Stellen vorgenommen werden.

> Besonders gravierend wird die Situation, weil es eine Vielzahl von Systembetreibern und eine stets wachsende Zahl von Anwendungsbereichen gibt. All diese Systembetreiber (z.B. Amtsstellen von Bund, Kantone und Gemeinden, Netzbetreiber, SBB, private Firmen) sind auf den korrekten Austausch von Daten angewiesen. Es macht aber wenig Sinn, wenn sie sich alle mit all den verschiedenen technischen Fragen befassen müssen. Der Aufwand ist erheblich. Zudem darf das Fehlerrisiko nicht unterschätzt werden. (weiter aus [**rahmenvorst-interlisopen**])

Da für verschiedene Anwendungsbereiche unterschiedliche Modelle notwendig sind, entsteht ein beträchtlicher Aufwand. Erschwerend kommen Probleme bei Beschaffung und Betrieb von Konversions-Systemen oder die Unterstützung von Partnern, welche nicht mit Interlis umgehen können, hinzu.

Vision Langfristig sollen sich die beteiligten Fachstellen um ihr Fachgebiet kümmern können, anstatt viel Energie für die technischen Details aufwenden zu müssen.

1.2 Ziele

Ein Grossteil der Diskussionen in der Schweizer Geo-Szene dreht sich um Format- und Implementations-Details, statt sich mit Modell- oder Fach-Fragen zu beschäftigen [**sfkeller**]. Um Bewegung in diese Situation zu bringen, soll ein Prototyp einer Konvertierungs- und Transformations-Plattform implementiert werden. Dieser dient als Demonstrationsobjekt für spätere Weiterentwicklungen und soll aufzeigen, wie eine solche Plattform aussehen könnte und wo allfällige Stolpersteine zu finden sind.

Projekt dat Es ist wichtig anzumerken, dass der ursprüngliche Titel dieser Arbeit "Project dat HSR" lautete und sich mit der Dokumentation und der Erweiterung des Projekts dat[1] befassen sollte. Dies wurde während der Arbeit aufgrund mangelnder Reife von dat abgebrochen und ist im **??** dokumentiert.

1.3 Rahmenbedingungen

Folgende Rahmenbedingungen wurden durch den Betreuer [**sfkeller**] vorgegeben:

- Python als Programmiersprache
- Postgres oder SQLite als Datenbank
- Deployment der Applikation muss via Web Server Gateway Interface (WSGI) möglich sein

Diese sind zugleich als nicht-funktionale Anforderungen zu verstehen.

[1] http://dat-data.com/

2 Umsetzung

2.1 Stand der Technik

In diesem Abschnitt werden bereits existierende Datenaustauschplattformen und Schematransformations-Tools vorgestellt.

2.1.1 dat

Das "dat"-Projekt will den Bezug und Austausch von Daten vereinfachen. Zusätzlich zum Datenspeicher wird dazu ein Synchronisierungsprotokoll, angelehnt an Replikationsprotokolle von bekannten Datenbanksystemen, verwendet.

Leider befindet sich das Projekt noch in den Kinderschuhen. Aus diesem Grund wurde beschlossen, Projekt dat nicht zu verwenden.

Für eine detailliertere Erläuterung, siehe **??**.

2.1.2 CKAN

Comprehensive Knowledge Archive Network (CKAN) ist ein Software-System zur Datenverwaltung, insbesondere von öffentlichen Daten. Schwerpunkte sind das Anbieten, Indexieren und Verwalten von Daten, welche von Benutzern zur Verfügung gestellt werden. Dahingehend kann eine grosse Menge an Meta-Informationen zusammen mit den Daten erfasst werden.

CKAN hat keine Unterstützung für Daten-Konvertierung oder -Transformation.

Leider wurden wir erst relativ spät auf CKAN aufmerksam. Daher wurde zusammen mit dem Betreuer beschlossen, bei dieser Arbeit den Schwerpunkt auf die Konvertierung und Transformation der Daten zu legen und allenfalls in einer späteren Arbeit CKAN und OpenDataHub zu integrieren.

opendata.admin.ch Open Data-Portal des Bundes, basierend auf CKAN (**??**).



Abbildung 2.1: Ein Daten-Paket auf opendata.admin.ch

datahub.io Datahub.io ist eine CKAN-basierte Plattform zum Anbieten von Daten (**??**).

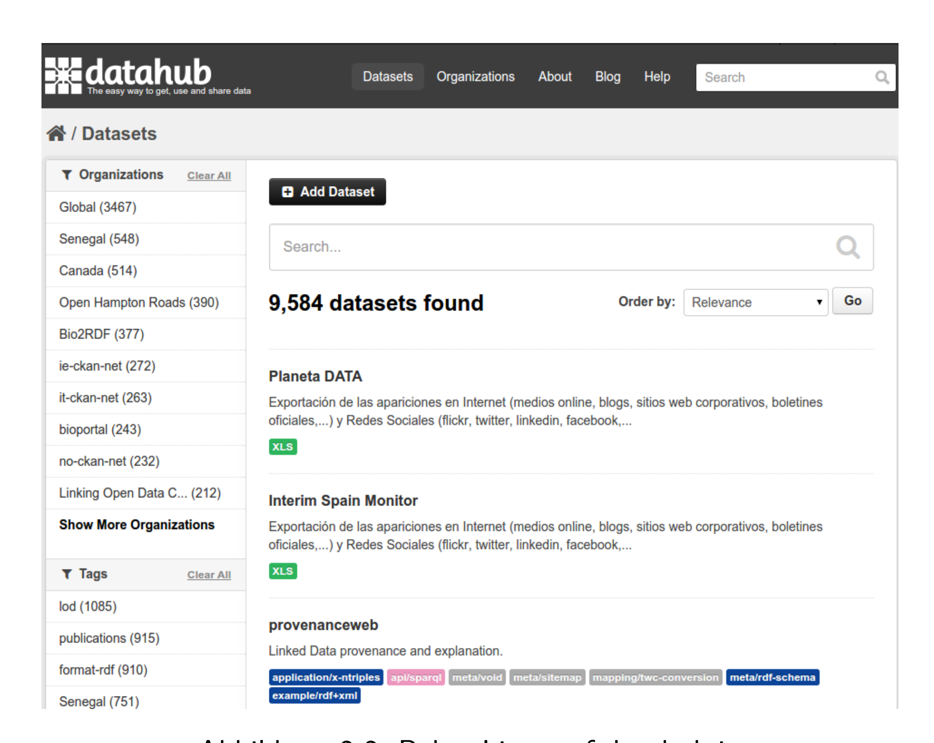


Abbildung 2.2: Paket-Liste auf datahub.io

opendatahub.it opendatahub.it ist eine italienische, ebenfalls CKAN-basierte Plattform.

2.1.3 Feature Manipulation Engine (FME)

Feature Manipulation Engine (FME) ist ein kommerzielles Produkt der Safe Software Inc. Der Fokus liegt auf der Transformation, Konvertierung und Integration von Daten in einer Desktop-Applikation.

geopol.ch Geopol ist eine kommerzielle Web-to-FME Schnittstelle.

2.1.4 Pentaho Kettle

Pentaho Kettle ist eine Open Source Desktop-Applikation für ETL-Aufgaben (**??**).

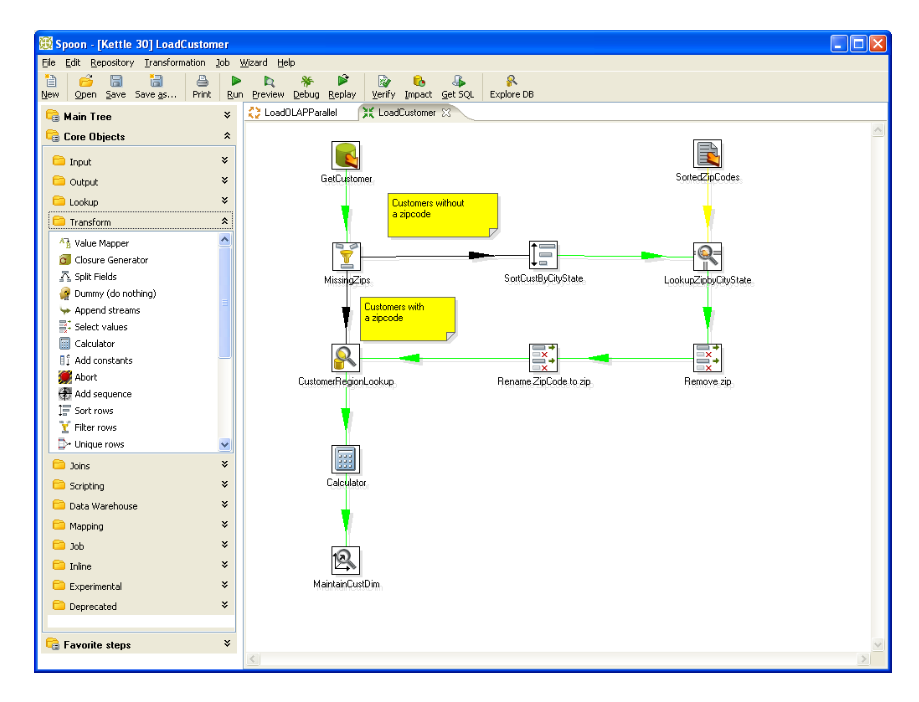


Abbildung 2.3: Transformations-Beispiel aus der Pentaho Kettle-Dokumentation

Mit GeoKettle[1] existiert eine Geo-Erweiterung. Diese wird jedoch angeblich nicht weiter entwickelt [**sfkeller**].

2.1.5 Yahoo! Pipes

Pipes erlaubt das Verknüpfen und Filtern von RSS-Daten. Interessant ist vor allem das User Interface, welches die grafische Verknüpfung von Datenquellen erlaubt (**??**).

[1] http://www.geokettle.org/

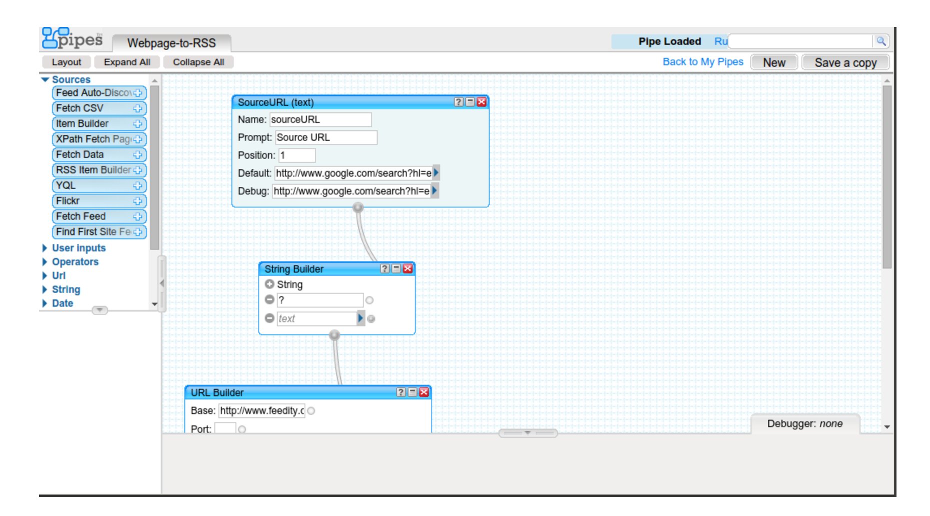


Abbildung 2.4: Pipe in Yahoo! Pipes

Yahoo! hat am 4. Juni 2015 angekündigt, dass Pipes am 30. September 2015 abgeschaltet wird.

2.1.6 OpenRefine.org

OpenRefine, früher bekannt als Google Refine, ist eine Open Source-Plattform zur Bereinigung und Transformation von Daten.

2.2 Konversion und Transformation

In diesem Abschnitt wird das grobe Umsetzungskonzept von OpenDataHub (ODH), in **??** ersichtlich, erläutert.

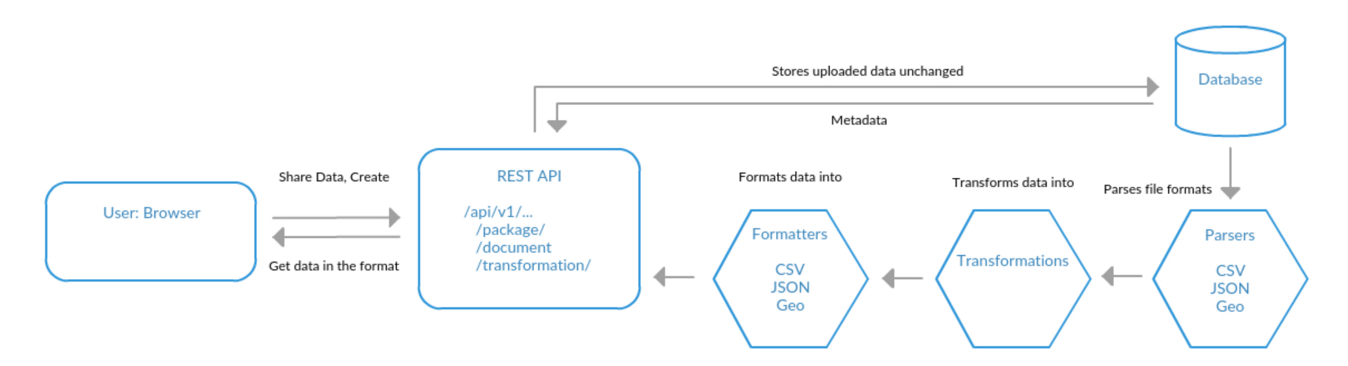


Abbildung 2.5: Grobe Architektur-Übersicht

2.2.1 Internes Format

Es wurden diverse Optionen sowohl zur Speicherung wie auch zur Transformation der Daten in Betracht gezogen und im Hinblick auf deren Vor- und Nachteile bezüglich Machbarkeit, Komplexität und Performance evaluiert. Die Daten werden von Parser-Modulen in sogenannte DataFrame Objekte

geladen. Ein DataFrame ist ein Tabellen-artiges Objekt der Pandas[2] Python-Bibliothek. Pandas ist eine mächtige "Data-Analysis" Bibliothek, welche aufgrund der zugrunde liegenden NumPy[3] Bibliothek sehr effizient ist. Zu jedem Format kann auch ein Formatter-Modul implementiert werden, welches ein DataFrame entgegennimmt und einen Output des gewünschten Formates zurückgibt.

2.2.2 Schema-Transformation

Auch bei der Homogenisierung der Schemata wurden diverse Möglichkeiten, welche teilweise eng mit dem Format gekoppelt[4] sind, analysiert. Schlussendlich wurde eine eigene Domain Specific Language (DSL) implementiert – die OpenDataHub Query Language (ODHQL). ODHQL ist ein Subset von SQL, inklusive Unterstützung einiger Geo-Basisfunktionalität analog der PostgreSQL-Erweiterung PostGIS.

2.2.3 Erweiterbarkeit

Sowohl die Unterstützung von Dateiformaten[5] wie auch die Funktionen der OpenDataHub Query Language wurden mit Bedacht auf Erweiterbarkeit implementiert. Eine Drittpartei kann durch Implementation eines speziellen Interfaces eigene Parser/Formatter Paare oder gar ODHQL Funktionen zur Verfügung stellen, welche nach einem Neustart der Applikation sofort verfügbar sind.

[2] http://pandas.pydata.org/
[3] http://www.numpy.org/
[4] Die Persistierung der Daten in eine Postgres Datenbank impliziert beispielsweise die Verwendung von PostgreSQL zur Transformation
[5] CSV, KML, Excel, ...

3 Resultate und Ausblick

3.1 Zielerreichung

Mit Ausnahme der Unterstützung des Formats GeoPackage wurden alle in der Aufgabenstellung gestellten Ziele erreicht. Der Grund für die mangelnde GeoPackage-Unterstützung ist eine Serie von Fehlern in Geospatial Data Abstraction Library (GDAL) (siehe **??**).

Einige selbst gesteckte Ziele wie z.B. "Integration SuisseID" konnten wir aufgrund sich verändernder Prioritäten nicht erreichen.

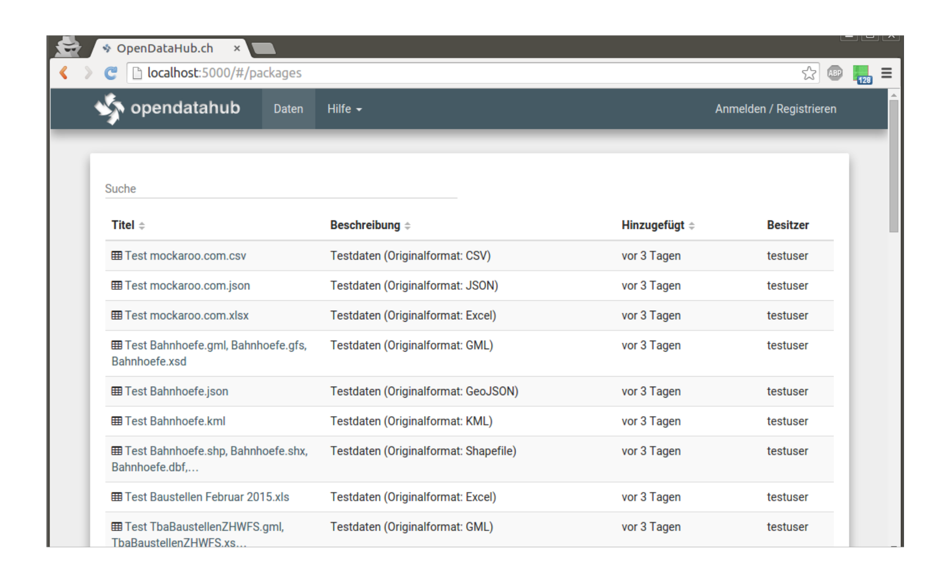


Abbildung 3.1: ODH: Paket-Liste

Ein Benutzer kann bereits vorhandene Daten durchsuchen und herunterladen. (**????**)

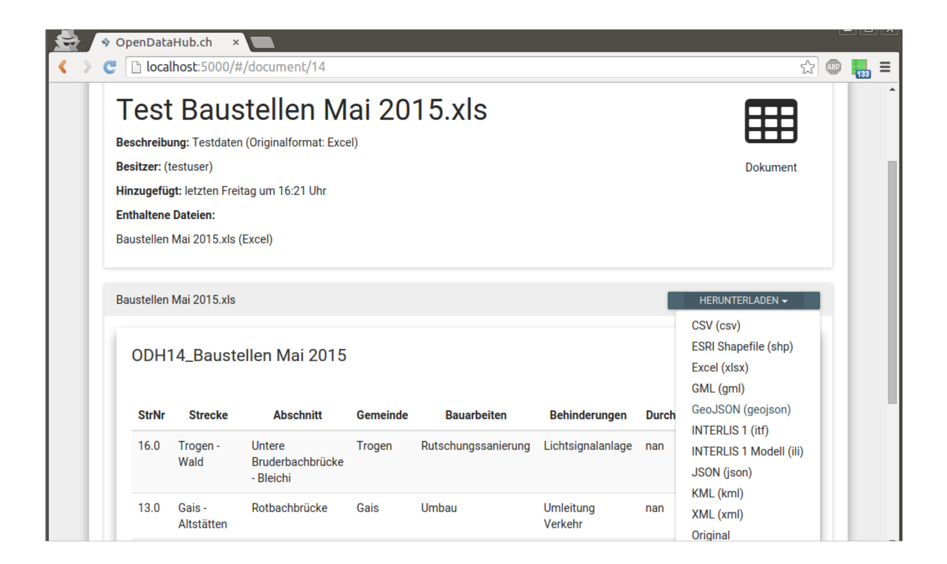


Abbildung 3.2: ODH: Detail zu hochgeladenen Daten

Daten können als Datei oder Gruppe von Dateien hochgeladen oder von einer URL bezogen werden. (**??**)

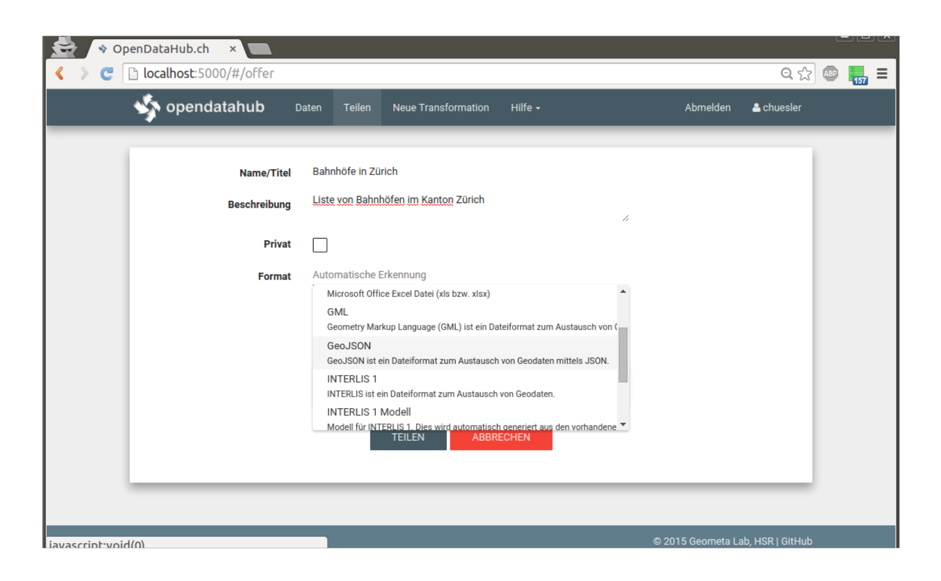


Abbildung 3.3: ODH: Daten anbieten

Die Resultate von Transformationen präsentieren sich ähnlich wie hochgeladene Dateien. (**??**)

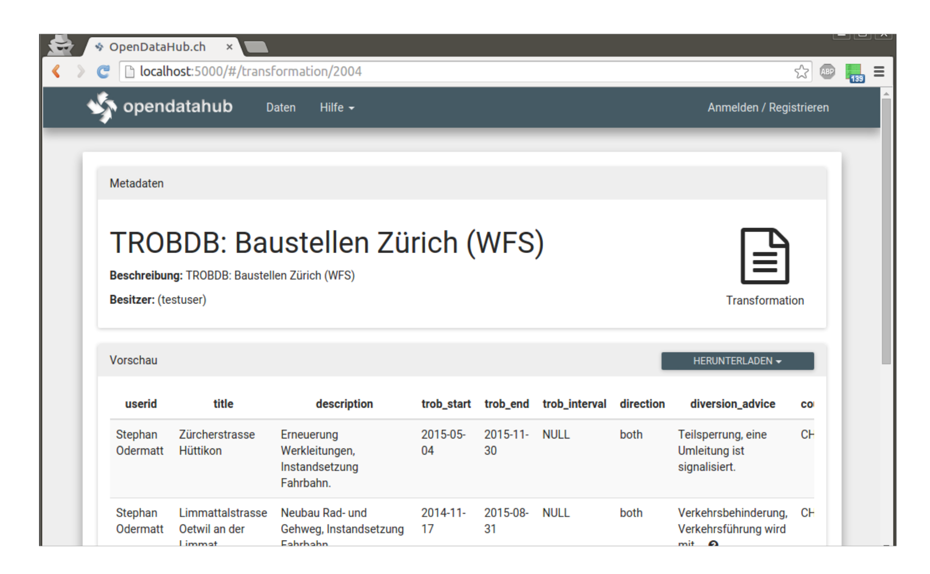


Abbildung 3.4: ODH: Details einer Transformation (1)

Mit minimaler Einführung können einfache Transformationen zusammen-geklickt werden (**??**). Mit ODHQL-Kenntnissen können auch komplexe Transformationen geschrieben werden. Als Datenquellen dienen hochgeladene Daten, WFS-Server oder bereits bestehende Transformationen.

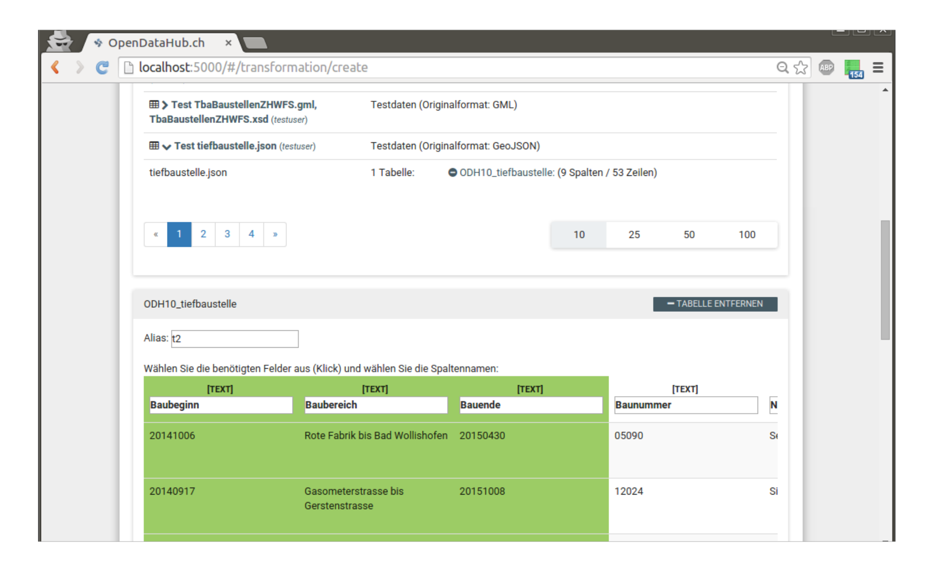


Abbildung 3.5: ODH: Transformations-Assistent

Bereits vorhandene Transformationen können geklont oder falls der eingeloggte Benutzer der Besitzer ist auch bearbeitet werden. (**??**)

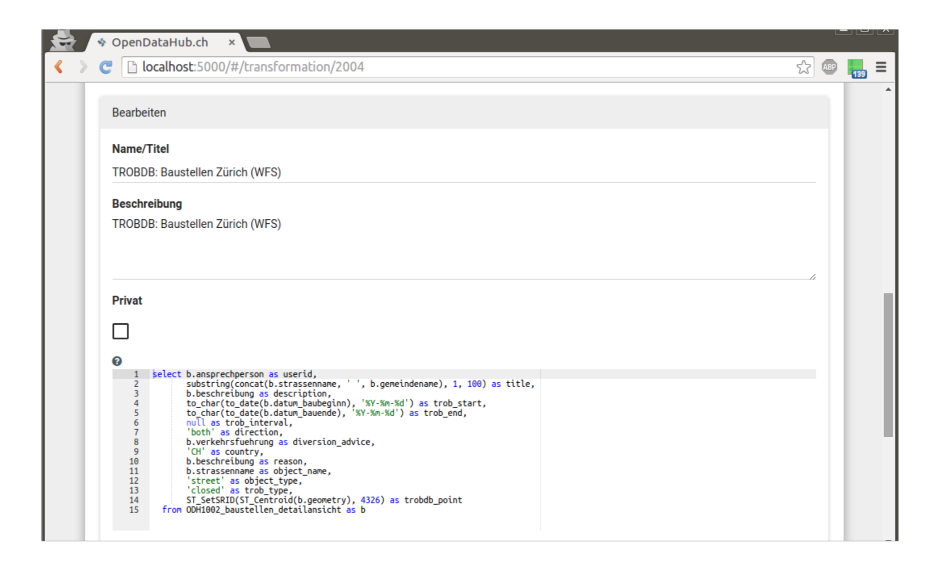


Abbildung 3.6: ODH: Details einer Transformation (2)

3.2 Ausblick: Weiterentwicklung

Für eine Weiterentwicklung sollten insbesondere folgende Punkte in Betracht gezogen werden:

- Wechsel auf GDAL Version 2.0 (sobald verfügbar): Aufgrund von Problemen mit mehreren Format-Treibern wurde für diese Arbeit eine etwas ältere Version eingesetzt (siehe **??**). Version 2.0 bringt diverse Verbesserungen sowie Treiber für neue Formate mit.

- Ausbau der Webapplikation bzw. Integration von CKAN: Die aktuelle Web-Applikation ist relativ einfach gehalten. So fehlt zum Beispiel die Möglichkeit, weitere Dateien zu einem Paket hinzuzufügen, oder welche zu entfernen.

- Ausbau des Transformations-Assistenten: Für einfache Transformationen ist der Assistent gut geeignet. Sobald allerdings Funktionen verwendet werden sollen, muss die Transformation direkt in ODHQL geschrieben werden.

Für weitere Ideen, siehe **??**.

3.3 Persönlicher Bericht

3.3.1 Christoph Hüsler

Meine Python-Vorkenntnisse waren relativ beschränkt - ich kannte die Syntax von Code-Beispielen, die mir über die Jahre im Internet begegnet sind und ich hatte bereits ein paar kleinere Skripts angepasst, aber nie ein eigenes Projekt mit Python realisiert. Trotzdem gelang es mir schnell, mich einzuarbeiten.

Highlight dieser Arbeit war für mich sicher die Implementation des ODHQL-Parsers mit PyParsing[1] - ich kann dieses Modul nur empfehlen.
Auch die Zusammenarbeit im Team empfand ich als sehr positiv. Trotz einiger kleiner Differenzen konnten wir immer miteinander diskutieren und eine Lösung finden.

3.3.2 Remo Liebi

Im Rahmen dieses Projektes habe ich mich vor allem um das Front-End gekümmert. Ich war vorher mit Web-Diensten vertraut, habe allerdings noch nie mit AngularJS gearbeitet. Es gelang mir dank der Hilfe von Fabio Scala und Christoph Hüsler schnell, mich in diesem JavaScript Framework einzuarbeiten und zurechtzufinden.
Meine Studienarbeit habe ich grösstenteils in Python geschrieben. Dadurch kenne ich mich in dieser Sprache gut aus und konnte mich auch schnell in das Django Framework einarbeiten.
Highlight dieser Arbeit war für mich die Implementation des Assistenten mit Hilfe von AngularJS. Ich konnte meine Ideen direkt umsetzen und habe viel über AngularJS gelernt.
Die Zusammenarbeit im Team empfand ich als angenehm und konstruktiv. Wir konnten uns immer gut austauschen, unsere Differenzen beseitigen und vom Know-how der anderen profitieren.

3.3.3 Fabio Scala

Trotz Hindernisse und Startschwierigkeiten zu Beginn der Arbeit, insbesondere durch das unreife "Projekt dat" sowie den natürlichen Differenzen im Team, ist es uns gelungen diese zu überwinden und eine vernünftige und moderne Webapplikation zu implementieren. Aus diesem Grund kann ich die Bachelorarbeit als positive und lehrreiche Erfahrung einstufen. Ich bin stolz auf das Team OpenDataHub und bedanke mich hiermit nochmals persönlich bei allen Beteiligten.
Bezüglich Technologien bin ich froh, dass ich als langjähriger Python Entwickler endlich Django Erfahrungen sammeln sowie auf Frontend-Seite die mächtige Kombination von AngularJS und TypeScript einsetzen konnte.

3.4 Dank

In erster Linie bedanken wir uns bei unserem Betreuer, Prof. Stefan Keller, für die Unterstützung während der gesamten Laufzeit der Arbeit. Zudem gilt auch Herrn Dorfschmid von Adasys AG ein besonderer Dank für das konstruktive Gespräch und die architektonische Inspiration für OpenDataHub. Weiter bedanken wir uns bei Herrn Kalberer, Entwickler von ogrtools und Interlis Treiber für GDAL sowie der GDAL Community für die stets rasche Rückmeldung auf Fehlerberichte unsererseits.
Auch unseren Freunden und Bekannten die unsere Arbeit getestet, kommentiert und korrigiert haben, sprechen wir unseren Dank aus.

[1] https://pyparsing.wikispaces.com/

Teil II

Projektdokumentation

1 Vision

Die Vision von OpenDataHub ist in **??**, **??** beschrieben.

2 Anforderungen

In diesem Kapitel sind die Anforderungen an OpenDataHub in Form von Scrum Epics und User Stories sowie weitere, nicht-funktionale Anforderungen dokumentiert.

2.1 User Stories

Die User Stories wurden aufgrund der zunächst unbekannten bzw. relativ knapp ausgedrückten Anforderungen iterativ aufgenommen. In **??** ist eine Kurzübersicht der Anforderungen als Use-Case Diagramm dargestellt und dient lediglich für Aussenstehende zum Verständnis. Die Akteure werden in **??** genauer erläutert.

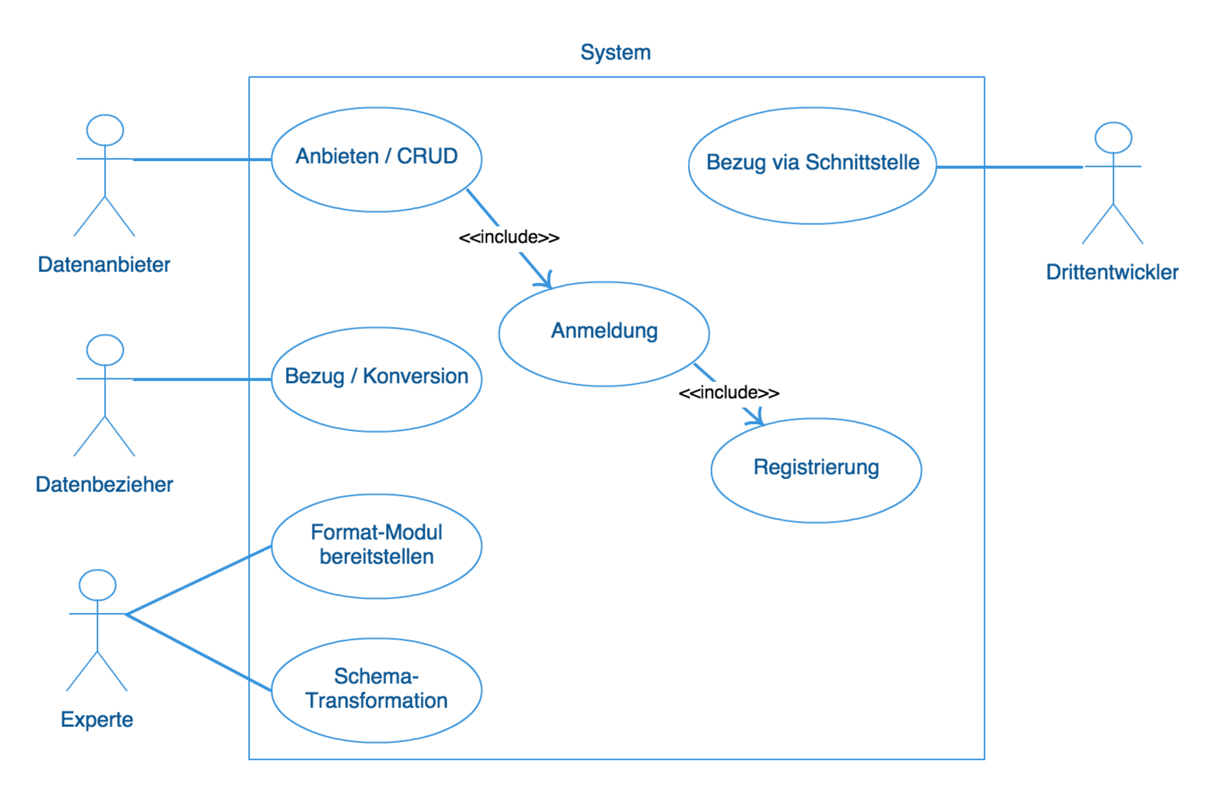


Abbildung 2.1: Übersicht Anforderungen

2.1.1 Konkrete Anwendungsfälle

Post- und Gebäudeadressen

Adressen im Allgemeinen können auf unterschiedliche Art und Weise modelliert und abgespeichert werden. Das klassische Beispiel ist die Adresse inklusive Hausnummer gegenüber zwei separaten

Feldern. Ziel ist es unterschiedliche Schemata von Postadressen als integriertes zu CSV generieren, oder vorhandene Gebäudeadressen mit Daten im MOpublic[1] Format anreichern zu können.

Verkehrshindernisse

Die Webapplikation Traffic Obstruction Database (TROBDB) stellt Verkehrshindernisse der Schweiz auf einer Karte dar. Diese Verkehrshindernisse werden von diversen Datenlieferanten in unterschiedlichen Datei-Formaten und Schemata angeliefert. Ziel ist es, alle diese Verkehrshindernisse in einem einheitlichen Datei-Format und Schema beziehen zu können.
Als Datenquellen dienen Datenangebote des Bundes, der Kantone sowie Liechtenstein. Als Demonstration sollen folgende Datenquellen verwendet werden:

- Daten von www.truckinfo.ch in einem nicht weiter spezifizierten XML-Format (siehe **??**)
- WFS des Kantons Zürich (Tiefbaustellen, Baustellen)
- Excel-Datei des Kantons Appenzell Ausserrhoden
- Für Google Earth erstelltes KML des Kantons Zürich

Programmcode 2.1: XML von truckinfo.ch

```
<evts>
<evt df="1430449200" dd="1430157600" r="A1" p="Switzerland" t="2" c="517" lar="3.7"
 ↪  cat="1_38" s="-1" x="539007" y="5257837" i="mapserver2/symbols/icone35.png">
[Switzerland] contraflow |Zurich - St. Margrethen|between junction St.
 ↪  Gallen-Kreuzbleiche and junction St. Gallen-Neudorf in both directions contraflow,
 ↪  temporary width limit 3.7 metres, set of roadworks during the night, length of the
 ↪  route affected: 3.7 km, speed limit: 80 km/h, duration: April 27, 2015 08:00 pm
 ↪  until May 01, 2015 05:00 am|
</evt>
<evt df="1430794800" dd="1430762400" r="A1" p="Switzerland" t="2" c="503" lar="6"
 ↪  cat="1_38" s="-1" x="533526" y="5257414" i="mapserver2/symbols/icone35.png">
[Switzerland] left lane(s) closed|St. Gallen - St. Margrethen|between junction St.
 ↪  Gallen-St. Fiden and junction St. Gallen-Neudorf in both directions left lane
 ↪  closed, temporary width limit 6.0 metres, set of roadworks during the night,
 ↪  length of the route affected: 1.3 km, speed limit: 80 km/h, duration: May 04, 2015
 ↪  08:00 pm until May 05, 2015 05:00 am|
</evt>
<!-- ... -->
</evts>
```

2.1.2 Rollen

In **??** sind die für das Projekt relevanten Rollen bzw. Akteure aufgelistet. Diese werden in den nachfolgenden Abschnitten referenziert.

[1] http://www.cadastre.ch/internet/cadastre/en/home/products/mopublic.html

Rolle	Beschreibung
Datenanbieter	Ist diejenige Person oder gar Unternehmen/Behörde, die daran interessiert ist, eigene Daten zur Verfügung zu stellen ohne sich um dessen Format oder Schnittstellen zum Bezug kümmern zu müssen.
Schema-/Format-Experte	Ist oftmals ein Datenanbieter selbst bzw. eine Person innerhalb dessen Organisation. Kann Schema-Transformationen mit OpenDataHub durchführen und ist dafür zuständig dem Administrator von OpenDataHub allfällige Module zur Unterstützung deren Format(e) innerhalb der Applikation zu liefern.
Datenbezieher	Der Datenbezieher ist lediglich an einzelnen angebotenen Daten interessiert und möchte diese für eigene Zwecke weiterverwenden.
Drittentwickler	Andere Entwickler, welche die Funktionalität der Applikation via REST API nutzen wollen.
Administrator	Der Applikationsverantwortliche, welcher die Applikation betreibt und wartet.

Tabelle 2.2: Rollen/Akteure

2.1.3 Registrierung / Authentifizierung

Epic 1: Benutzer registrieren

Als Datenanbieter will ich mich in der Applikation registrieren können, sodass ich Daten unter meiner Person/Organisation publizieren kann.

User Story 1.1: SuisseID

Als Datenanbieter will ich mich mittels SuisseID registrieren/anmelden können, sodass ich als Behörde einen Zugriff erlange ohne einen Social-Account zu benötigen.

User Story 1.2: GitHub

Als Datenanbieter oder Drittentwickler will ich mich mittels GitHub-Konto registrieren/anmelden können, sodass ich Daten publizieren kann ohne ein separates Konto zu benötigen.

User Story 1.3: Facebook

Als Datenanbieter oder Drittentwickler will ich mich mittels Facebook-Konto registrieren/anmelden können, sodass ich Daten publizieren kann ohne ein separates Konto zu benötigen.

2.1.4 Daten publizieren

Epic 2: Daten publizieren

Als Datenanbieter will ich die Daten in diversen strukturierten Formaten anbieten können, sodass ich diese der Öffentlichkeit ohne zusätzlichen Aufwand bereitstellen kann.

Akzeptanzkriterien
Gegeben
– Eine oder mehrere Dateien in einem von der Applikation unterstützten Format,
wenn der Datenanbieter die Datei via Webapplikation publiziert, dann soll diese in OpenDataHub abgespeichert werden und in einer Liste von verfügbaren Daten ersichtlich sein.

User Story 2.1: Datei hochladen

Als Datenanbieter will ich eine oder mehrere Dateien in einem strukturierten Format via Browser hochladen können, sodass ich diese auf der Plattform weiterverwenden kann.

User Story 2.2: Online Quellen

Als Datenanbieter will ich Daten in einem strukturierten Format mit Angabe einer Online-Quelle (HTTP, WFS) publizieren können, sodass ich diese auf der Plattform weiterverwenden kann.

Akzeptanzkriterien
Gegeben
– Daten sind Online in einem von der Applikation unterstützten Format verfügbar,
wenn ich die Daten publiziere, dann sollen diese von der Applikation automatisch aus der angegebenen Quelle in die Applikation geladen werden und zur Verfügung stehen.

User Story 2.3: Private Daten

Als Datenanbieter will ich Daten als “Privat” markieren können, sodass diese nur von mir ersichtlich und beziehbar sind und nicht der Öffentlichkeit zur Verfügung stehen.

Akzeptanzkriterien
Gegeben
– Das Formular zur Publikation von Daten,
wenn ich den Eintrag als Privat markiere, dann sollen die Daten für anonyme oder andere Benutzer nicht (mehr) in der Liste von verfügbaren Daten gelistet sein und jeglicher Zugriff darauf verweigert werden.

User Story 2.4: Modul anbieten

Als Datenanbieter bzw. Schema/Format-Experte will ich Python-Module (Plugins) welche eine gegebene Schnittstelle implementieren, anbieten, um die bereitgestellten Daten/Formate unterstützen zu können.

Akzeptanzkriterien
Gegeben

- Einer vordefinierten Schnittstelle zur Programmierung eines solchen Moduls,
- einem Beispiel eines solchen Moduls

wenn ein solches Modul bereitgestellt wird, dann soll das Format von der Applikation unterstützt werden.

2.1.5 Daten transformieren

Epic 3: Transformation

Als Experte will ich Transformationen mittels einer vorgegebenen Sprache oder Konfiguration erstellen können, sodass diverse heterogene Daten homogenisiert und in einem einheitlichen Format bezogen werden können.

Akzeptanzkriterien
Gegeben

- Einer dokumentierten Schema-Mapping-Sprache oder -Konfiguration,
- einem Beispiel eines solchen Mappings/Transformation,

wenn ich eine solche Transformation an die Applikation übergebe, dann soll diese auf die ausgewählten Daten angewendet werden. Das Resultat will ich analog wie einzelne Daten in **??** in diversen Formaten beziehen können.

User Story 3.1: Transformation publizieren

Als Experte will ich diese Transformationen ("Views" auf die Daten) analog wie die einzelnen Daten speichern und publizieren können, sodass diese einerseits immer wieder per Knopfdruck angewendet und andererseits auch von anderen, nicht-Experten verwendet werden können.

2.1.6 Daten beziehen

Epic 4: Datenbezug

Als Endbenutzer will ich die bereitgestellten Daten in einem von mir gewünschten, strukturierten Format beziehen können.

Akzeptanzkriterien
Gegeben

- Eine in der Applikation von einem Datenanbieter abgelegten Datei,
- eine Datei dessen Format von der Applikation unterstützt wird,

wenn ich die Datei mittels Browser in einem anderen Format beziehen will, dann soll die automatisch ins Zielformat konvertierte Datei heruntergeladen werden.

2.2 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen der einzelnen User Stories sind grundsätzlich in deren Akzeptanzkriterien enthalten. In diesem Abschnitt werden zusätzliche, aus der Aufgabenstellung entnommene sowie bei dem Betreuer aufgenommene nicht-funktionale Anforderungen, ergänzt durch sinnvolle Anforderungen moderner Software-Entwicklung beschrieben.

2.2.1 Technologien

Die Anforderungen an die eingesetzten Technologien[2] sind teilweise durch die Aufgabenstellung sowie durch die bereits existierende Infrastruktur am Geometa Lab der Hochschule für Technik Rapperswil (HSR) wie folgt eingeschränkt:

- Python als Programmiersprache
- Deployment der Applikation muss via WSGI möglich sein
- PostgreSQL als Datenbank oder SQLite bei kleineren Aufgaben

Die Wahl der Frameworks und Libraries, sowohl Server- wie auch Client-seitig ist dem Entwicklerteam überlassen.

2.2.2 Qualität

Zur Sicherung der Softwarequalität wurden folgende drei Anforderungen durch den Betreuer bestimmt:

- Einsatz von automatisierten Unit Tests
- Kommentieren des Codes in einem Sphinx[3] kompatiblen Format
- PEP-8 als Codierrichtlinie, besonders die Verwendung von Leerzeichen zur Einrückung

[2] Programmiersprachen, Frameworks, Libraries, Services, ...
[3] Siehe auch: http://sphinx-doc.org

2.2.3 Effizienz

Bei der Applikation handelt es sich einerseits um einen Prototypen, andererseits nicht um eine Echtzeit-Applikation. Dennoch sollte die Applikation mit Datenmengen von maximal 500000 Datensätzen umgehen können, sofern genügend Arbeitsspeicher vorhanden ist.

3 Project dat

3.1 Einführung

In der heutigen Gesellschaft wird der Austausch von Daten immer wichtiger. Vermehrt werden grosse Datenmengen unterschiedlichster Art - aus Forschung, Regierung oder zivilen Kreisen - zur allgemeinen Verwendung angeboten. Nicht nur die Art der Daten ist jedoch vielfältig, sondern auch Datenformat oder Zugriffsart unterscheiden sich.
Das dat-Projekt hat das Ziel, die Integration sowie einen einheitlichen Zugriff solcher Datenquellen zu vereinfachen.
Als erstes Ziel dieser Arbeit sollte das noch junge Projekt dat anlysiert und dessen genaue Funktionalität und Funktionsweise dokumentiert werden. Aufgrund der Tatsache, dass dat noch relativ neu ist und sich in der Alphaphase befindet, ist das Projekt nur wenig und sehr verstreut dokumentiert. Dat hat im Jahr 2014 einen Gesamtbetrag von $310'000 von der Knight Foundation sowie der Sloan Foundation zur Weiterentwicklung des Projekts erhalten, was als Indiz für ein vielversprechendes Projekt gilt.

3.2 Was ist dat?

Das dat-Projekt hat folgende Ziele [**what-is-dat**]:

- Daten sollen automatisch zwischen unterschiedlichen dat-Instanzen synchronisiert werden können.
- Unterstützung grosser Datenmengen[1], evtl. mit häufigen Aktualisierungen
- Unterstützung von tabellarischen oder unstrukturierten Daten
- Plugin-basierte Schnittstelle zu bestehenden Datenbanken/Formaten
- Unterstützung für automatisierte Workflows

Für den Umgang mit verschiedenen Datenformaten können Transformationen [**dat-js-api**] definiert werden, welche entweder vor Schreib- oder nach Lese-Operationen ausgeführt werden.

[1] Milliarden von Datensätzen bzw. Speicherbedarf im TB-Bereich

3.3 Architektur

dat speichert Daten auf zwei verschiedene Arten:

- Für Tabellen-Daten wird LevelDB verwendet. Da LevelDB ein austauschbares Backend hat, können diese Daten auch in einer SQL-Datenbank wie Postgres gespeichert werden.

- BLOB-Daten werden standardmässig im File-System per local-blob-store gespeichert. Dies kann ebenfalls durch eine kompatible Implementation ausgetauscht werden.

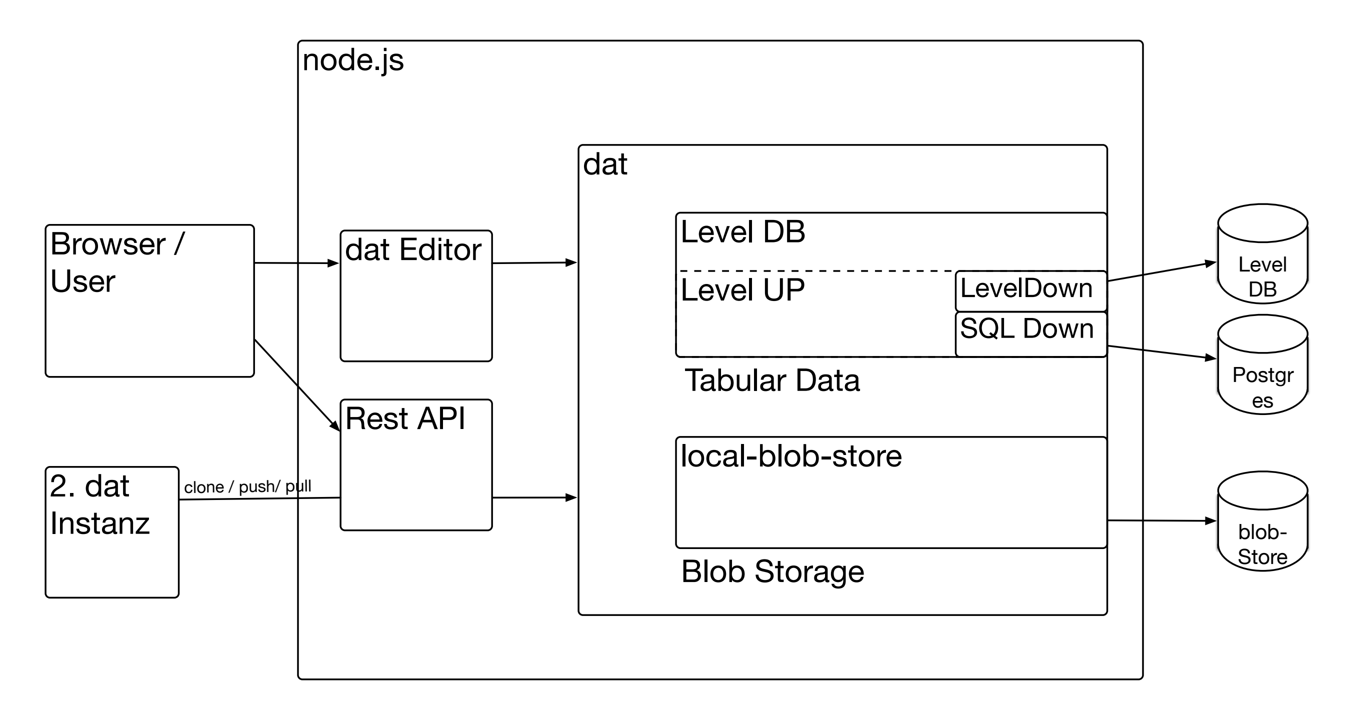


Abbildung 3.1: dat Architektur-Übersicht

Direkt mit dat wird auch der dat-editor geliefert, welcher als rudimentäres Web-GUI dient. Der Editor sowie das REST-API werden gestartet, wenn `dat listen` aufgerufen wird. Andere dat-Instanzen verwenden das REST-API für die Synchronisation.

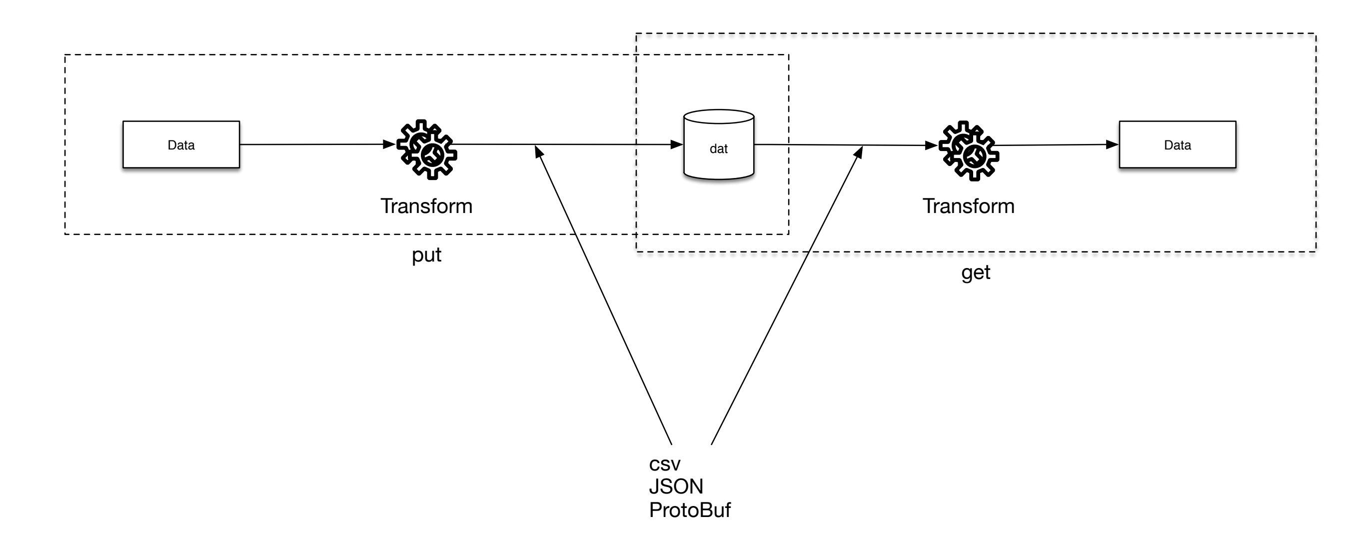


Abbildung 3.2: dat Daten-Pipelines

Mithilfe des Sub-Projekts `gasket` können sogenannte Daten-Pipelines definiert werden. Diese dienen dazu, Daten abzurufen, zu transformieren und schliesslich in dat zu speichern, bzw. Daten aus dat zu lesen, zu transformieren und schliesslich abzuliefern.

3.4 Use Cases

In diesem Abschnitt werden mögliche Verwendungszwecke von dat aufgezeigt.

3.4.1 Astronomie – Trillian

In der Astronomie fallen riesige Datenmengen an. Teilweise werden diese als grosse Daten-Releases zur Verfügung gestellt (z.B. Sloan Digital Sky Survey), welche frühere Releases komplett ersetzen. Andere Projekte stellen inkrementelle Updates zur Verfügung (z.B. Hubble). Viele Astronomie-Institute haben weder die Mittel noch das Know-how, mit solchen Datenmengen umgehen zu können. Das Trillian-Projekt kümmert sich um die Verwaltung dieser Daten und bietet eine Compute-Engine an, um neue Modelle anhand der vorhandenen Daten zu testen.
dat soll für den Import und die Indexierung der Daten verwendet werden.
Siehe auch https://github.com/maxogden/dat/issues/172.

3.4.2 Regierung – Sammlung von Daten

Für Statistik- oder Regulierungszwecke sammeln Regierungen Daten von unterschiedlichsten Betrieben. Diese Daten müssen oft über ein mehr oder weniger brauchbares Portal abgeliefert werden.
Die Verwendung von dat bringt folgende Vorteile:

- Ermöglicht die Prüfung von Daten beim Import, bereits vor der Ablieferung der Daten
- Ermöglicht Ergänzung oder Korrektur von bereits abgelieferten Daten.

- Ersatz der bisher verwendeten, für jede Regierungsstelle neu kreierten und teuren Portale durch eine standardisierte Lösung.

Siehe auch https://github.com/maxogden/dat/issues/153.

3.5 CLI

Das dat Command-line interface (CLI) “`dat`” ist das einzige Werkzeug um ohne API mit dem Repository zu interagieren und bietet folgende Basisfunktionalität:

- Erstellung neuer dat Repositories/Intanzen mit `$ dat init`
- Import von CSV oder JSON Dateien mit `$ dat import`
- dat Server und Webfrontend starten mit `$ dat listen`
- Klonen eines kompletten[2] remote Repositories dat Repostitories analog Git mit `$ dat clone`
- Lokale Änderungen ebenfalls analog Git wieder zurück ins remote Repository synchronisieren mit `$ dat push`
- Rudimentäre Manipulation und Ausgabe von einzelnen Zeilen des Repositories mit `$ dat rows` bzw. `$ dat cat`

Die abschliessende Liste von zurzeit unterstützten Befehlen ist in **??** zu finden. Für fortgeschrittenere Verwendung ist das CLI jedoch nicht geeignet, da viele Optionen fehlen. Stattdessen sollte das JavaScript-API verwendet werden.
Spätere Versionen sollen auch Branches erlauben, sowie mit Änderungskonflikten umgehen können.

3.6 Frontend

Dat bietet ein minimales Webfrontend welches ähnliche Operationen wie mit dem CLI wie z.B. Import/Export sowie das Betrachten der Datensätze ermöglicht. Einen Ausschnitt ist folgender **??** zu sehen.

[2] Samt aller Historie

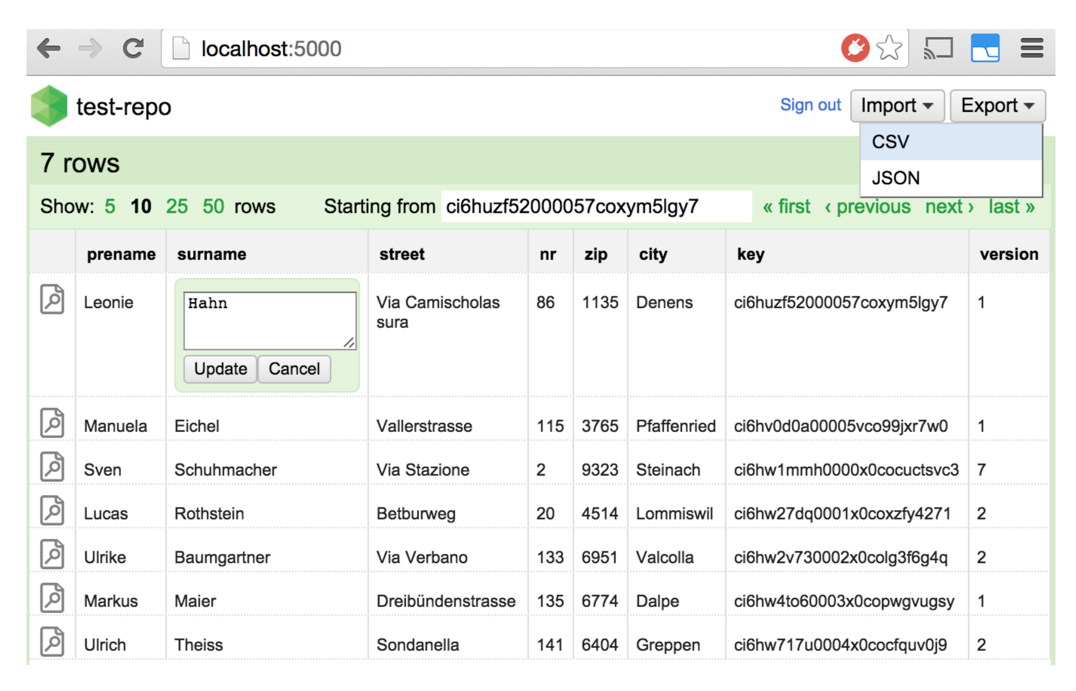


Abbildung 3.3: dat Webfrontend

3.7 Schnittstellen

Für eine Beschreibung der von dat zur Verfügung gestellten Application Programming Interface (API), siehe **??**.

3.8 Fazit

Zum aktuellen Zeitpunkt (Februar 2015) implementiert dat alpha 6.9.6 folgende Features:

- Datenspeicher auf Basis eines Key-/Value-Stores (LevelDB).
- Limitierte Synchronisation basierend auf einem Change-Stream. So weit wir erkennen können ist keine Konflikt-Verwaltung vorhanden, sondern lediglich Ein-Weg-Synchronisation vorgesehen.
- Pro dat-Instanz existiert nur ein Schema. Falls ein Datensatz mit neuen Spalten hinzugefügt werden soll, wird das bereits vorhandene Schema mit dem Schema des neuen Datensatzes zusammengeführt.

Aktuell läuft die Entwicklung der dat beta-Version. In dieser sollen mehrere “datasets” unterstützt sowie bessere Versionskontrolle mit Konfliktbehandlung bzw. -Vermeidung implementiert werden. Leider ist diese Version noch nicht stabil und das API ändert sich damit komplett.

Entscheid 3.1: Projekt dat

Aufgrund der Tatsache, dass dat, obschon vielversprechend, noch sehr unausgereift und undokumentiert ist, sowie das API noch ständigen Änderungen unterzogen wird, wurde nach Rücksprache mit dem Betreuer gegen den Einsatz von dat entschieden. Das Projektrisiko ist zu gross und es wird nach einer alternativen Lösungen zur Implementation eines Datenhubs gesucht.

Anmerkung: Bei Abschluss der Arbeit ist dat ist nun in Version 6.9.7[3] (vom 25. Februar 2015) verfügbar.

[3] siehe https://github.com/maxogden/dat/releases

4 Analyse

4.1 Domäne

Die Applikation besitzt, wenn man die Daten selbst nicht miteinbezieht, ein relativ simples Domänen-Modell, wie in **??** zu sehen ist. Der Endbenutzer kann mehrere Dateien ablegen oder bereits vorhandene transformieren. Da das Resultat von Dateien oder Transformationen dasselbe ist, werden diese mit dem Oberbegriff Datenpaket[1] versehen.

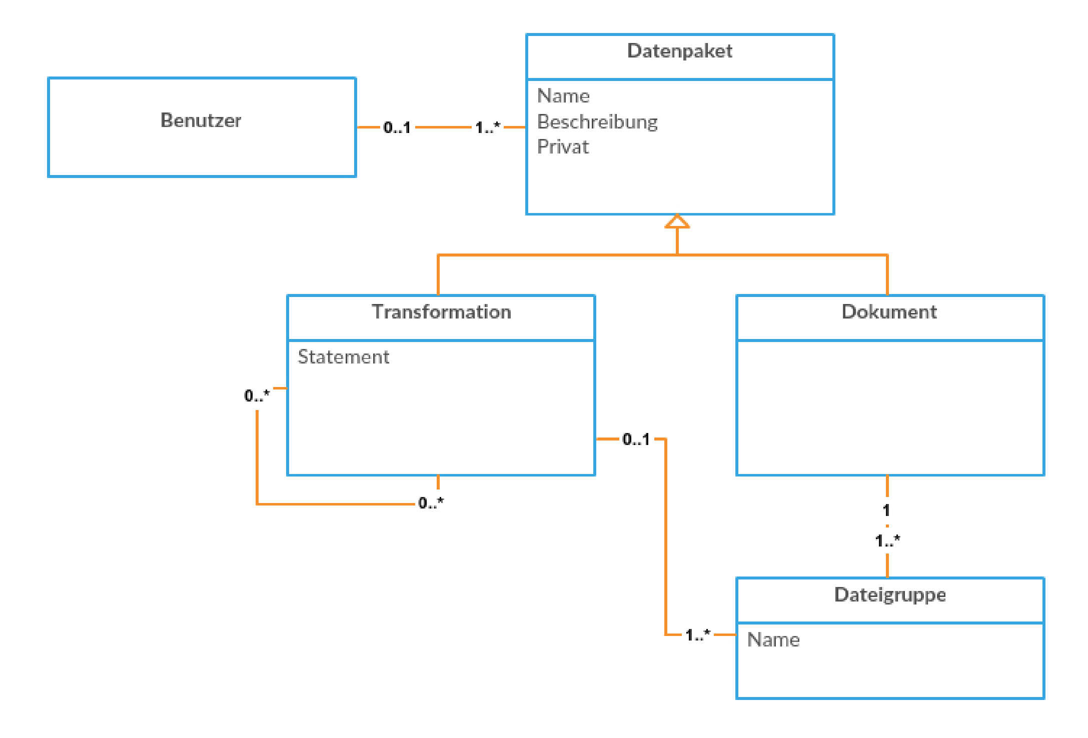


Abbildung 4.1: Domäne OpenDataHub

4.2 Dateiformate

In diesem Abschnitt werden die durch den Kunden gewünschten oder zur Realisierung der Anwendungsfälle in **??** benötigten Dateiformate und deren Eigenheiten beschrieben.

[1] Im Frontend: "Package"

4.2.1 CSV

Das Comma Separated Values (CSV) gilt trotz vielen Schwächen als universelles Austausch/Exportformat für einfache Anwendungsfälle und sollte deshalb zwingend unterstützt werden. Eine CSV Datei enthält einen Datensatz pro Zeile und die Spalten üblicherweise durch Kommas oder Semikolon getrennt. Die erste Zeile enthält die Namen der einzelnen Felder.

Erweiterung mit Typen und Geo-Daten

GeoCSV ist eine durch das Geometa-Lab der HSR spezifizierte Erweiterung von CSV um Geo-Daten mit folgenden Eigenheiten [**sfkeller**, **geocsv**, **gdal-csv**]:

- Eine optionale Datei mit Endung `.csvt` spezifiziert auf einer einzigen Zeile die verwendeten Datentypen
- Den spezielle Datentyp "WKT" für Geometrien im Well-Known Text (WKT) Format
- Die speziellen Datentypen "Point(x)", "Point(y)" oder "CoordX" bzw. "CoordY" für den Spezialfall einer Punktgeometrie
- Eine optionale Datei mit der Endung `.prj` zur Angabe des Referenzsystems, ebenfalls im WKT Format[2]
- Nur ein Geometrie-Objekt/Spalte pro Datei

4.2.2 Excel

Da viele nicht-technische Benutzer mit Microsoft Excel arbeiten [**sfkeller**] oder Daten im Excel Format anliefern, müssen sowohl das alte Excel 2003 (xls) sowie das neue OpenXML (xlsx) Format unterstützt werden.

4.2.3 GeoJSON

GeoJSON ist eine auf JSON basierende Schema-Spezifikation für Geo-Daten. Dieses Format ist heutzutage weit verbreitet und wird fast überall, wo mit Geo-Daten hantiert wird, unterstützt.

4.2.4 GeoPackage

GeoPackage ist ein relativ neues, auf SQLite basiertes, Format. Es ist sehr nahe an SpatiaLite, welches mit Version 4.2 auf GeoPackage wechselt.

4.2.5 GML

Geometry Markup Language (GML) ist ein auf XML basierendes Format für Geo-Daten und ist das Standard-Ausgabeformat von WFS-Servern.

[2] Achtung, das `.prf` Format von ESRI sieht sehr ähnlich aus, enthält jedoch weniger Informationen!

4.2.6 Interlis

Interlis besteht aus einer Modellbeschreibungssprache und einem Transferformat. Da es praktisch nur in der Schweiz zum Einsatz kommt, ist die Unterstützung in kommerzieller Software mässig bis nicht vorhanden.
Interlis existiert in zwei Versionen: Interlis 1 verwendet ein Datensatz-basiertes Transfer-Format, während für Interlis 2 XML gewählt wurde. Mit wenigen Ausnahmen stellt Interlis 2 eine Erweiterung von Interlis 1 um Objekt-orientierte Features dar.

4.2.7 KML

Keyhole Markup Language (KML) ist wie GML ebenfalls ein auf XML basierendes Format, welches in Google Earth Verwendung findet. Speziell bei KML ist, dass es lediglich das Referenzsystem 4326 (klassisches lat/lon) unterstützt.

4.2.8 ESRI Shapefile

Das Shapefile ist ein von der Firma ESRI spezifiziertes Format zum Austausch von Geo-Daten. Es wird von fast allen Bibliotheken und Systemen unterstützt und daher oft als universelles Austauschformat für Geo-Daten verwendet. Das Shapefile wird immer wieder aufgrund diverser Schwächen kritisiert, z.B. die Limitation der Spaltennamen auf 8 Zeichen sowie von Textspalten auf 255 Zeichen. Aus diesem Grund wird versucht, das veraltete Shapefile durch ein moderneres Format wie GeoJSON oder GeoPackage abzulösen [**sfkeller**].

4.2.9 WFS

Bei Web Feature Service (WFS) handelt es sich um einen speziellen Webservice für Geo-Daten. Die Daten können üblicherweise im GML Format bezogen werden. Da es sich dabei um einen weit verbreiteten Standard des Open Geospatial Consortium (OGC) handelt, sollte die Applikation einen direkten Bezug von Daten via einem WFS Server ermöglichen.

4.2.10 XML

Da es sich bei XML um eine potenziell verschachtelte Datenstruktur handelt, wird vorerst nur eine flache Version unterstützt, ähnlich der von der API von truckinfo.ch gelieferten und in **??** beschriebenen Daten.

4.2.11 JSON

Auch bei JSON handelt es sich um ein Format mit Schachtelungsmöglichkeit. Aus diesem Grund wird auch hier vorerst nur die oberste Ebene betrachtet.

4.3 Datenspeicherung

Ein Ziel dieser Arbeit ist die Erstellung eines Datenhubs, welcher es Anbietern ermöglicht, ihre Daten anderen zur Verfügung zu stellen.
Zur Speicherung der Daten stehen folgende Varianten zur Auswahl.

4.3.1 File-basiert

Die Daten werden direkt im Dateisystem gespeichert. Um Konflikte zwischen Dateinamen zu vermeiden muss ein Hashing-Schema definiert und verwendet werden. Metadaten zu den Dokumenten werden in einer Datenbank hinterlegt.
Die Daten werden in ihrem Ursprungsformat belassen, bis ein Nutzer eine Transformation auswählt und anwendet. Die Resultate einer solchen Transformation können direkt wieder im Dateisystem zwischengespeichert werden, um weitere Abfragen nach der selben Transformation zu beschleunigen.
Die Tatsache, dass die Daten im Dateisystem liegen, vereinfacht die Einbindung von Software von Drittanbietern, insbesondere wenn diese Software nur direkt auf Dateiebene arbeitet (z.B. OGR).
Ein Nachteil dieser Variante ist, dass Cloud-Anbieter wie Heroku keine beschreibbaren [**herokuReadOnlyFilesyste** bzw. keine beständigen [**herokuEphemeralFilesystem**] Dateisysteme anbieten, sondern auf die Verwendung von Amazon S3 oder ähnlich verweisen.
Metadaten wie Format oder Original-Dateinamen würden aus Performance-Gründen weiterhin in einer Datenbank gehalten werden.

4.3.2 Datenbank-basiert, einzelne Zeilen

Die Daten werden in einzelne Zeilen zerlegt und zusammen mit den Metadaten in einer Datenbank gespeichert. Dies erfordert, dass bereits der Daten-Anbieter eine Transformation in das Zeilen-Format auswählen muss. Falls sich herausstellt, dass dies zu langsam ist um es direkt im HTTP-Request des Anbieters durchzuführen, müssen die Daten vor der Transformation zwischengespeichert werden.
Falls die Transformationsresultate zwischengespeichert werden, sollen ist eine weitere Datenablage für die Transformationsresultate notwendig.
Die Einbindung von Drittanbieter-Software erfordert mehr Aufwand, falls diese nur auf Dokumenten- oder gar Datei-Ebene arbeiten. In diesem Fall müssen die Daten erst wieder in ein geeignetes Format übersetzt werden, um nach der Transformation wieder in einzelne Zeilen zerlegt zu werden.

4.3.3 Datenbank-basiert, Dokument als BLOB

Die Daten werden ohne Transformation zusammen mit den Metadaten in einer Datenbank abgelegt.
Diese Variante hat praktisch alle Vorteile der datei-basierten Version.
Falls Drittanbieter-Software nur auf Dateibasis arbeitet kann ein temporäres oder flüchtiges Datei-System verwendet werden.
Für diese Variante existieren folgende Möglichkeiten:

Daten und Metadaten in derselben Tabelle Daten und Metadaten werden in der selben Tabelle gespeichert. Dies bietet ein einfacheres Schema, erlaubt jedoch nur ein Daten-Objekt pro Dokument, was ungeeignet ist für Dokument-Typen wie ESRI Shapefiles, welche aus mehr als einer Datei bestehen.

Daten und Metadaten in separaten Tabellen Daten und Metadaten werden in unterschiedlichen Tabellen gespeichert. Dies erlaubt die Speicherung von Dokumenten bestehend aus mehreren Dateien, oder transformierten Daten als zusätzlichen Daten-Eintrag zu einem Metadaten-Eintrag. Dies erlaubt es auch, Django-Models zu erstellen, welche nicht jedes mal die Daten abfragen, obwohl nur Metadaten benötigt werden.

Da Dateiformate existieren, welche aus mehr als einer Datei bestehen, ist die Option mit separaten Tabellen vorzuziehen.

Fazit

?? erläutert die Kriterien für die Bewertung der verschiedenen Varianten.

Kriterium	Beschreibung	Gewichtung
Original bleibt erhalten	Das Originaldokument bleibt erhalten oder kann komplett rekonstruiert werden. Dies bietet folgende Vorteile: Verbesserungen an der Software (Parser) zeigt auch bei alten Daten Wirkung, ausserdem gehen hierbei keine Daten verloren.	★★★
Einbindung Software	Erlaubt die Einbindung von Drittanbieter-Software. Dies ist möglich auf Dateibasis, oder direkt als Memory Buffer, falls Python-Code aufgerufen wird.	★☆☆
Beliebige Formate	Die Variante unterstützt beliebige Formate. Anschliessende Transformationen können nur auf bekannte Formate angewandt werden, aber auch unbekannte Formate können gespeichert und zum Download angeboten werden. Die Daten können verwendet werden, falls später ein Parser für das bisher unbekannte Format geschrieben wird .	★★☆
Cloud	Cloud-Anbieter wie Heroku bieten oft kein permanentes File-System, sondern verweisen auf Storage Provider wie Amazon S3 oder auf Datenbanken als Daten-Speicher. Unterstützung für Cloud-Anbieter ist kein zwingendes Kriterium für diese Arbeit, soll aber falls möglich erhalten bleiben.	★☆☆

Tabelle 4.2: Bewertungskriterien Datenspeicherung

?? fasst die verschiedenen Optionen zusammen.

Kriterium	File	DB: Records	DB: Schemalos
Original bleibt erhalten	★★★	★☆☆	★★★
Einbindung Software	★★★	★★☆	★★★
Beliebige Formate	★★★	★☆☆	★★★
Cloud	★★☆	★★★	★★★
Total	★★★	★☆☆	★★★

Tabelle 4.4: Auswertung verschiedener Speicher-Varianten

Entscheid 4.1: Datenspeicherung

Aus Gründen der Kompatibilität zu Cloud-Anbietern, insbesondere Heroku, ohne gleich Cloud-Datenspeicher wie Amazon S3 verwenden zu müssen, wurde bereits früh entscheiden, die Dateien in der Datenbank zu persistieren. Da die Datenhaltung ohnehin eine sehr lose Kopplung zur restlichen Applikation besitzen soll, kann künftig auch eine andere Strategie implementiert werden.

4.4 Intermediäres Format und Transformationssprache

4.4.1 Motivation und Voraussetzungen

Dateiformat

Bei N Dateiformaten würden bei 1-zu-1 Format-Konversionen bereits $\frac{N \cdot (N-1)}{2}$ Übersetzungsmodule benötigt, um lediglich zwischen den Dateiformaten zu konvertieren. Zudem muss es möglich sein, das Schema der Daten zu transformieren. Diese Tatsache erfordert zwingend ein intermediäres, einheitliches Format, ohne welches eine generische Transformation unmöglich wäre.

Transformationssprache

Gleichzeitig kann die Wahl eines intermediären Formates nicht unabhängig von der Wahl einer Transformations-Implementation getroffen werden. So wäre es zum Beispiel unpraktikabel, als Zwischenformat eine Datenbank-basierte Lösung zu wählen, während die Transformation auf Dateiebene stattfinden soll.

Für unterschiedliche Datenstrukturen sind verschiedene Transformationsansätze geeignet. Diese Arbeit befasst sich aus Zeitgründen nur mit tabellarischen Daten.

?? fasst die Anforderungen an eine Transformationssprache zusammen.

Anforderung	Beschreibung
Mapping	Felder müssen 1:1 auf andere Felder zugeordnet werden können (Feld umbenennen)
Default-Werte	Neue Felder mit Default-Werten müssen hinzugefügt werden können
Joins	Mehrere Datenquellen müssen miteinander verknüpft werden können. Dabei muss immer eine Join-Bedingung angegeben werden (keine kartesischen Produkte)
Filter/Prädikate	Daten sollen gefiltert werden können
Erweiterbare Operationen	Es sollen neue Operationen definiert werden können (vorzugsweise in Python)
Sortierung	Das Resultat der Transformation soll sortiert werden können

Tabelle 4.6: Anforderungen an die Transformationssprache/Konfiguration

4.4.2 Datenbank

Intermediäres Format

Bei dieser Variante wäre das intermediäre Format äquivalent der Datenspeicherung wie sie in **??** beschrieben ist. Die Daten würden aber dennoch separat als Originaldateien abgelegt werden um keine Informationen zu vernichten.

Transformationssprache

Vorteile dieser Variante sind, dass die Abfrage-Sprache bereits vorhanden und bekannt ist und dass je nach verwendeter Datenbank auf vorhandene Funktionen und Optimierungen zurückgegriffen werden kann. Falls Postgres mit PostGIS zum Einsatz kommt, kann auch im Geo-Bereich eine umfassende Funktions-Bibliothek verwendet werden.
Neue Funktionen müssen zwangsweise auf Datenbank-Ebene implementiert werden, bei Postgres z.B. mit Procedural Language/PostgreSQL Structured Query Language (PL/pgSQL). Diese Variante bringt praktisch alle verlangten Features ohne grossen Implementationsaufwand mit. Das Hinzufügen bisher nicht vorhandener Funktionalität ist jedoch eher schwierig.

Anmerkungen

Da die Format-Konversionen auf Python-Ebene durchgeführt werden, liegt folgender Datenfluss vor:

- Zu Beginn befinden sich die Daten als BLOB in der Datenbank.
- Um die Daten zu parsen werden sie mit Python ausgelesen.
- Der Parser schreibt die Daten als Records wieder in die Datenbank.

- Die Transformation findet statt. Aus Effizienzgründen könnte hier z.B. eine Materialized View[3] verwendet werden. Die Resultate werden von Python ausgelesen.
- Ein Formatter verwendet diese Daten, um das gewünschte Format zu erzeugen.

In diesem Prozess werden die Daten mehrfach aus der Datenbank gelesen, bzw. wieder in diese hineingeschrieben. Gleichzeitig kann die Datenbank viele ihrer Vorteile, wie z.B. Indizes nicht ausspielen, da sowieso meist alle Daten benötigt werden.
Diese Effekte führen dazu, dass massive Performance-Einbussen gegenüber anderen Optionen zu erwarten sind.

4.4.3 Ogrtools

Ogrtools ist eine Python-Implementation des CLI-Tools "ogr2ogr", erweitert mit simplen Transformationen. Ogr2ogr wie auch ogrtools verwenden die kompilierten OGR/GDAL Bibliotheken, welche 1-zu-1 (ogr → ogr) Format-Transformationen ermöglichen. Beispielsweise können ESRI Shapefiles (.shp) in KML oder GML Dateien umgewandelt werden. Da die Formate untereinander nicht immer kompatibel sind, kann es durchaus zum Verlust von Informationen kommen.

Intermediäres Format

Bei dieser Version befindet sich die komplette Parser → Transformation → Formatter-Kette in ogr2ogr/GDAL. Als Zwischenformat können die GDAL-internen OGR-Datenstrukturen betrachtet werden.

Transformationssprache

Ogrtools beinhaltet bereits eine Transformations-Konfiguration namens ogrtransform. Die Transformation selbst wird ebenfalls von GDAL durchgeführt. Eine ogrtools Beispielkonfiguration [**ogrtools**] ist in **??** zu sehen.

Programmcode 4.1: ogrtools Mapping-Konfiguration

```
{
  "//": "OGR transformation configuration",
  "src_format": "Interlis 2",
  "dst_format": "PostgreSQL",
  "dst_dsco": {},
  "dst_lco": {
    "SCHEMA": "public"
  },
  "layers": {
    "roadsexdm2ben_roads_streetnameposition": {
      "fields": {
        "tid": {
          "src": "TID",
          "type": "String"
        },
```

[3] http://www.postgresql.org/docs/9.3/static/sql-creatematerializedview.html

```
          "street": {
            "src": "Street",
            "type": "String"
          },
          "namori": {
            "src": "NamOri",
            "type": "Real"
          }
        },
        "geometry_type": "Point",
        "src_layer": "RoadsExdm2ben.Roads.StreetNamePosition",
        "geom_fields": {
          "nampos": {
            "src": "NamPos",
            "type": "Point"
          }
        }
      },
      "roadsexdm2ben_roads_streetaxis": {
        "fields": {
          "tid": {
            "src": "TID",
            "type": "String"
          },
          "street": {
            "src": "Street",
            "type": "String"
          }
        },
        "geometry_type": "MultiLineString",
        "src_layer": "RoadsExdm2ben.Roads.StreetAxis",
        "geom_fields": {
          "geometry": {
            "src": "Geometry",
            "type": "MultiLineString"
          }
        }
      }
    }
    ...
  }
```

Aktuell werden diverse der gewünschten Features von ogrtransform nicht unterstützt, wie z.B. Joins oder Filter. Sofern GDAL diese unterstützt, bedeutet das eine Erweiterung auf Python-Ebene. Andernfalls muss eine Erweiterung auf GDAL-Ebene durchgeführt werden.
GDAL unterstützt eine grosse Menge an Formaten[4]. Leider sind jedoch nicht alle Treiber von der selben Qualität, und Dateiformate ohne Geometrie sind schlecht vertreten.

[4] Sofern alle Features beim Kompilieren aktiv sind: 139 Raster- und 83 Vektor-Formate (Stand 6. Juni 2015)

4.4.4 Python Tabellenstrukturen

Intermediäres Format

Petl ist eine Python ETL Bibliothek, welche es erlaubt, diverse Formate in eine Tabellenstruktur zu laden und mit einem API darauf diverse Transformationen anzuwenden oder gar ganze Transformations-Pipelines aufzusetzen.
Der Vorteil von petl gegenüber anderen Lösungen liegt in dessen "lazy evaluation". Es werden keine Daten geladen oder transformiert, bis das Resultat benötigt wird. Dann erst werden die Daten Zeile für Zeile eingelesen und durch die Transformations-Pipeline geschleust. Petl kann somit mit sehr grossen Datenmengen umgehen, ist aber im Vergleich zur Konkurrenz langsamer und für Performance-kritische Applikationen ungeeignet [**petl**].

Pandas ist eine Bibliothek zur Datenanalyse, sozusagen das "R"[5] für Python. Doch die zugrunde liegenden, ebenfalls tabellenartigen Datenstrukturen zusammen mit der API von Pandas machen Transformationen wie Joins, Filtern, Umbenennungen und Berechnungen sowie String-Operationen sehr einfach. Pandas baut auf NumPy auf, der de-facto Standardbibliothek für Matrizen/Berechnungen in Python. Da NumPy in C bzw. Fortran und das Python API sehr funktional-orientiert implementiert sind, lassen sich Operationen auf ganze Spalten oder Tabellen anwenden, was zu einer für Python-Verhältnisse sehr hohen Performance führt [**pandasMergePerformance**].

Programmcode 4.2: Pandas mit GeoPandas

```
>>> import geopandas
>>> gdf = geopandas.GeoDataFrame.from_file('Bahnhoefe.shp')
>>> gdf.columns
Index([u'CNTRYNAME', u'CONURB', u'DUP_NAME', u'LEVEL', u'NAME', u'NATION',
 ↪   u'PROV1NAME', u'TYPE', u'geometry'], dtype='object')
}
>>> gdf.CONURB[:10]
0         None
1      Chiasso
2      Balerna
3    Mendrisio
4     Capolago
5     Capolago
6     Capolago
7     Capolago
8     Capolago
9     Maroggia
>>> gdf[gdf.CONURB == 'Zürich'][:5]
        CNTRYNAME  CONURB DUP_NAME  LEVEL                  NAME  NATION  \
1317  Switzerland  Zürich        N     10      Sood Oberleimbach      41
1346  Switzerland  Zürich        N     10        Zürich Leimbach      41
1360  Switzerland  Zürich        N     10         Zürich Mannegg      41
1373  Switzerland  Zürich        N     10     Zürich Wollishofen      41
1374  Switzerland  Zürich        N     10   Zürich Tiefenbrünnen      41
```

[5] http://www.r-project.org

```
     PROV1NAME  TYPE                                    geometry
1317    Zürich    30  POINT (681918.8961174991 241523.9264312923)
1346    Zürich    30  POINT (681750.8866669014 243289.5358952266)
1360    Zürich    30  POINT (681746.1775964648 244178.9977101207)
1373    Zürich    30  POINT (682765.2814791016 244804.6936600676)
1374    Zürich    30  POINT (685054.0860048197 244870.4588255175)
>>> gdf[gdf.geometry.within(shapely.geometry.Polygon([(704940,231559),
 ↪  (704949,231559),(704945,231550),]))]
        CNTRYNAME      CONURB DUP_NAME  LEVEL        NAME  NATION   PROV1NAME  \
1196  Switzerland  Rapperswil        N     10  Rapperswil      41  Ostschweiz

      TYPE                                    geometry
1196    30  POINT (704945.8093275279 231556.7192434487)
```

PyTables/HDF ist eine high-performance Python-Bibliothek die wie Pandas ebenfalls auf NumPy aufbaut. Da das API von PyTables im Vergleich zu Pandas eher low-level ist und weniger bietet wurde diese Bibliothek nicht weiter verfolgt.

Transformationssprache

DSL: Vereinfachtes SQL Dieses Format lehnt sich an SELECT aus SQL an. Es ist anzunehmen, dass Experten im Bereich Datentransformation sich mit SQL auskennen, was das Erlernen dieser Query-Sprache sehr vereinfachen würde [**sfkeller**].

Da als Backend nicht eine SQL-Datenbank verwendet wird, sondern die Operationen auf Pandas DataFrames übersetzt werden, sind einige kleinere Differenzen zu SQL unvermeidbar.

Neue Funktionen können in Python definiert werden.

DSL: Eigenes Format Diese Variante besitzt grundsätzlich dieselben Eigenschaften wie eine SQL-basierte DSL. Dieser gegenüber entfällt jedoch das Familiaritäts-Argument. Ausserdem muss eine passende, einfach zu erlernende Syntax definiert werden.

Aus diesen Gründen wird diese Variante nicht weiter verfolgt.

4.4.5 Fazit

?? erläutert die wichtigsten Kriterien für die Bewertung der beschriebenen Format-Lösungen.

Kriterium	Beschreibung	Gewichtung
Performance	Wie performant ist die Lösung? Wichtig aufgrund des Effizienz-NFRs, welches beim Treffen mit Herrn Dorfschmid geäussert wurde.	★★☆
Machbarkeit	Die generelle Umsetzbarkeit der funktionalen Anforderungen mit dieser Lösung.	★★★
Aufwand / Komplexität	Der benötigte Aufwand um Applikation der Lösung umzusetzen.	★★☆

Tabelle 4.8: Bewertungskriterien Intermediäres Format

Zusätzlich werden die in **??** beschriebenen Kriterien für die Bewertung der Transformationssprachen verwendet.

Features	Bereits vorhandene Unterstützung für die in **??** aufgelisteten Features	★☆☆
Geo-Daten	Unterstützung für Geo-Daten	★★★
Allgemeine Daten	Unterstützung für Nicht-Geo-Daten	★★★
Funktionen	Erweiterbarkeit um eigene bzw. benutzerdefinierte Funktionen (in externer Sprache)	★★☆
Familiarität	Wie gut kennen Experten auf dem Gebiet Datentransformation diese Sprache, bzw. wie viel Lernaufwand ist notwendig, um die Sprache zu erlernen	★☆☆

Tabelle 4.10: Bewertungskriterien Intermediäres Format/Transformationssprache

?? fasst die verschiedenen Optionen zusammen.

Kriterium	Datenbank	ogrtools	Petl	Pandas	PyTables
	SQL	ogrtransform	SQL-ähnliche DSL		
Performance	★☆☆	?	★☆☆	★★☆	★☆☆
Machbarkeit	★★☆	★☆☆	★★☆	★★★	★★☆
Aufwand / Komplexität	★★☆	★☆☆	★★☆	★★☆	★☆☆
Features	★★★	★☆☆		★★★	
Geo-Daten	★★★	★★★		★★★	
Allgemeine Daten	★★★	★☆☆		★★★	
Funktionen	★★☆	★☆☆		★★★	
Familiarität	★★★	★☆☆		★★★	
Total	★★☆	★☆☆	★★☆	★★★	★★☆

Tabelle 4.12: Auswertung verschiedener Format- und Transformations-Varianten

Entscheid 4.2: Intermediäres Format

Aufgrund der sowohl mächtigen API wie auch der gleichzeitig hohen Performance und Integration mit Geo-Daten bzw. Shapely wurde für Pandas und deren Datenstrukturen entschieden. Die Variante "Datenbank" wurde, obschon der Aufwand für die Transformationen weitgehend entfallen würde, aufgrund der Machbarkeit im Zusammenhang mit Echtzeit-Datenquellen verworfen.
Als Transformationssprache wird eine an SQL angelehnte DSL implementiert.

4.5 User Interface

4.5.1 Backend

Im Python-Umfeld existieren zwei für uns bekanntere Frameworks für Webapplikationen: Flask[6] und Django[7]. F. Scala und R. Liebi haben in ihren Semesterarbeiten Flask eingesetzt, und geraten, Django zu verwenden.
Um einfach ein REST API erstellen zu können wird zusätzlich die Django-Erweiterung rest_framework[8] verwendet.

Entscheid 4.3: Backend

Flask hat sich in früheren Arbeiten nicht empfohlen. Daher setzen wir in dieser Arbeit Django ein.

[6] http://flask.pocoo.org
[7] https://www.djangoproject.com
[8] http://www.django-rest-framework.org

4.5.2 Frontend

Technologie

AngularJS ist ein weit verbreitetes und etabliertes Frontend Framework basierend auf MVVM. Im Rahmen der Studienarbeiten wurde dies von Teilen unseres Teams bereits eingesetzt.

Entscheid 4.4: AngularJS als Frontendtechnologie

Da Christoph Hüsler und Fabio Scala bereits viel Erfahrung mit AngularJS sammeln konnten und es sich dabei um ein etabliertes Framework handelt, wurde für AngularJS und gegen andere potenziellen JavaScript Frameworks entschieden. Eine detaillierte Evaluation fand bereits im Rahmen der Studienarbeiten von Fabio Scala und Remo Liebi statt.

> Im Kontext der Evaluation eines für die Aufgabenstellung passenden Frontend JavaScript Frameworks wurde, um eine rapide Entwicklung mit geringen Support-Risiken zu erreichen für AngularJS und gegen Ember, Knockout, Backbone und andere entschieden und wir akzeptieren, dass der Einsatz mit gewissen Einarbeitungsaufwänden verbunden ist. [**poitour**]

> ... Es gibt eine grosse Anzahl solcher Bibliotheken, doch eine der populärsten ist sicherlich AngularJS. Angular basiert auf dem MVC Prinzip und bietet viele Vorteile zum interaktiven Bedienen einer Webseite. Somit haben wir in der SA auf Angular JS gesetzt ... [**wohnortfinder**]

Bootstrap

Bootstrap ist das ideale Framework um rasch einen funktionierenden Prototypen im Web zu entwerfen. Die Verwendung von Frontend-Frameworks beschleunigt die Entwicklung, da zu Beginn bereits klar ist, welche Komponenten vorhanden sind und wie diese aussehen. Die Integration von Bootstrap in AngularJS via Plugins vereinfacht die Verwendung der Frameworks enorm. Ein Theme steuert das Look and Feel von Bootstrap. Material Design for Bootstrap setzt die Design-Vorgaben für Material UI auf Android 5 mit Bootstrap um und garantiert dadurch ein modernes Look-and-Feel.

Entscheid 4.5: Bootstrap als Designgrundlage

Aufgrund der Vertrautheit mit Bootstrap haben wir uns für die Verwendung von Bootstrap entschieden. Als Theme wird Material Design for Bootstrap verwendet.

Menupunkte

Die Menuführung soll schlicht und übersichtlich gehalten werden. Der Prototyp wird folgende Struktur umfassen: Daten - Daten bereitstellen - Transformation erstellen.
Unter dem Menupunkt Daten werden bestehende Dokumente und Transformationen bereitgestellt.

Assistent

Es kann sich schwierig gestalten, eine Transformation von Grund auf zu erstellen. Deshalb soll eine Art Assistent zur Verfügung gestellt werden, welcher eine initiale ODHQL-Abfrage erstellen kann. Dieses soll auch Abfragen mit JOIN und UNION unterstützen.

Drafts

Zu Beginn des Projekts wurden einige Skizzen erstellt, wie das Front-End aussehen könnte. Einige davon werden in diesem Abschnitt aufgezeigt.

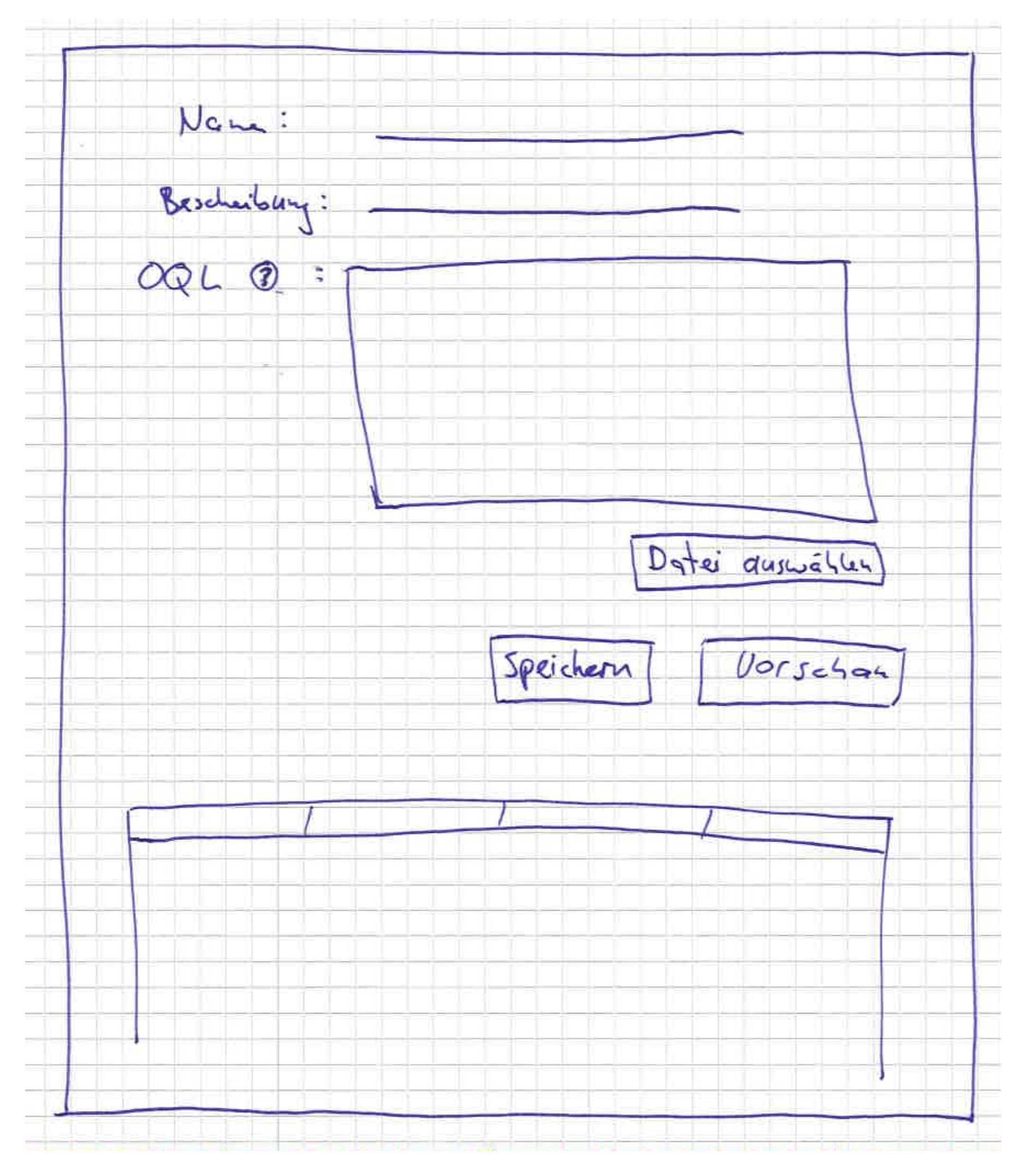


Abbildung 4.2: Frühes Mockup “Transformation erstellen”

Bevor klar wurde, dass es zur Erstellung von Transformationen einen Assistenten benötigt, war dies

wie in **??** angedacht. Auch einen grafischen Editor - analog wie bei Yahoo Pipes (eingestellt) - zu implementieren, war eine Option (siehe **??**).

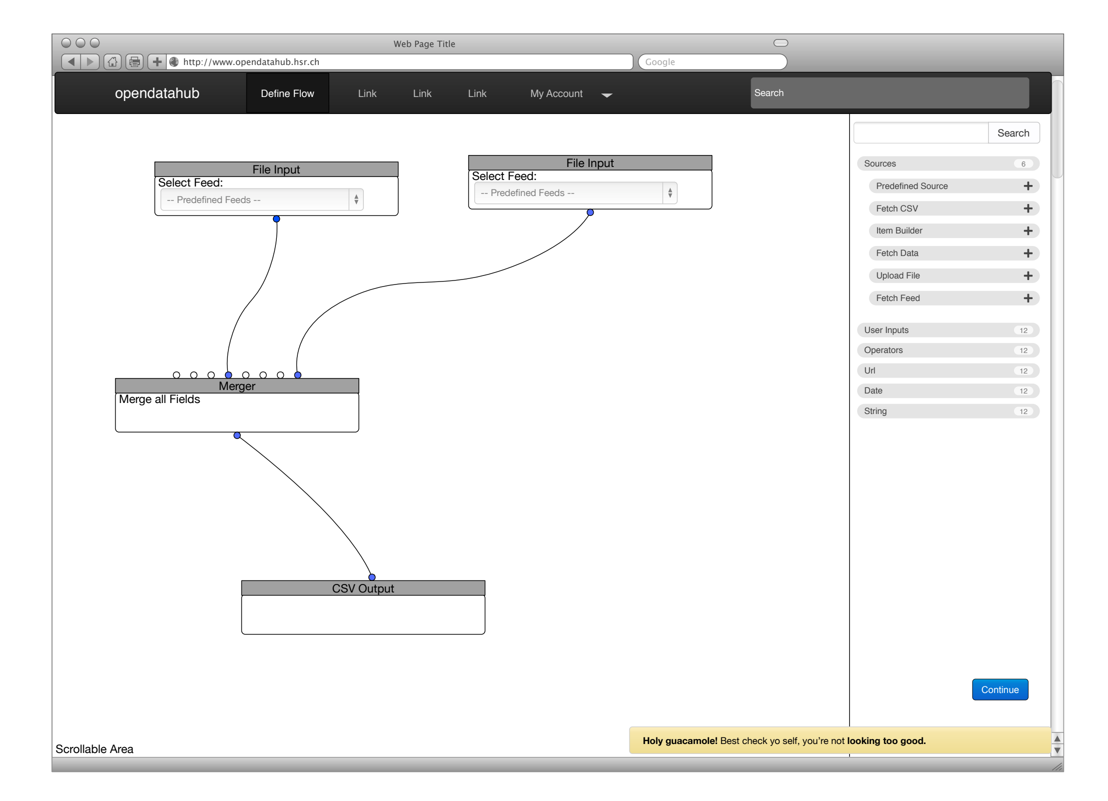


Abbildung 4.3: Frühe Idee Transformation mit "Pipes" erstellen. (Verworfen)

Das Upload-Formular wurde sehr nahe dem in **??** gezeigten Wireframe implementiert.

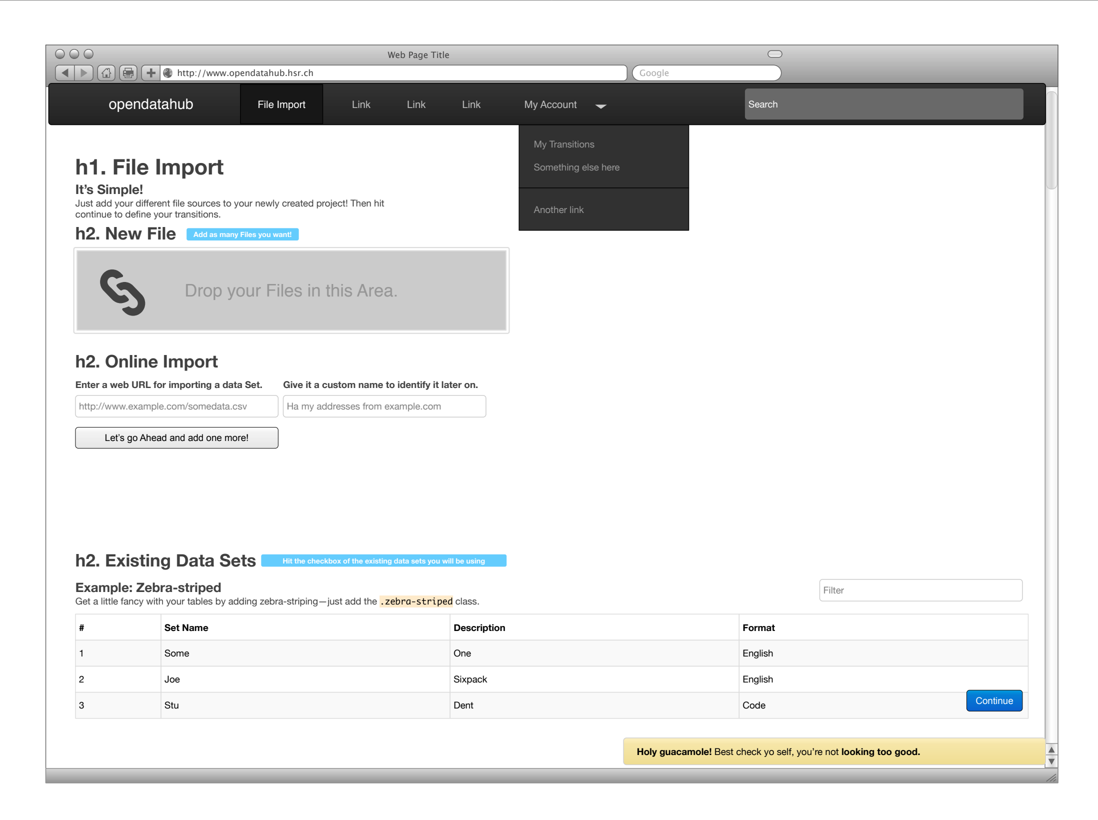


Abbildung 4.4: Frühes Mockup für Upload Formular

4.6 Benutzer-Authentifizierung

In diesem Kapitel werden verschiedene Varianten zur Authentifizierung erläutert.

4.6.1 Herkömmliche Registrierung per E-Mail

Die klassische Benutzer-Registrierung bietet den Vorteil, dass sie keine Abhängigkeiten zu Drittparteien besitzt. So kann sich jeder Benutzer mit einer gültigen E-Mail Adresse registrieren und anmelden. Ein Nachteil dieser Methode kann sein, dass Dummy-Adressen verwendet werden können und so ein registrierter Benutzer nicht wirklich authentifiziert ist. Nachteilig ist, dass der Benutzer nicht verifiziert ist. Benutzer müssen ein Passwort speziell für diesen Service erstellen um den persönlichen Sicherheitsansprüchen zu genügen.

4.6.2 OAuth2

OAuth2 hat einen anderen Ansatz. Soziale Netzwerke bieten sich an, um mit den Login-Daten des bereits bestehenden Accounts in einem solchen ein Account bei OpenDataHub zu erstellen. So ist der Benutzer der sich registriert bereits authentifiziert und es kann davon ausgegangen werden, dass es sich wirklich um diesen handelt. Die Häufigkeit von"Fake-Profilen" wird dadurch reduziert. Auch muss sich der Benutzer kein weiteres Passwort merken.

4.6.3 Fazit

Obwohl sich diese Methoden verbinden liessen, setzen wir einzig und alleine auf die OAuth Methode.

Entscheid 4.6: Benutzer-Authentifizierung

Aufgrund der Entscheidung, das Benutzer-Handling einfach zu halten, haben wir uns dazu entschieden, nur auf die Methoden zur Authentifizierung per GitHub und Facebook zu basieren.

4.7 Session-Authentifizierung

4.7.1 Cookie-Based-Authentication

Bei der Cookie-Based-Authentifizierung wird eine Session-ID auf der Seite des Clients in einem Server-Side Cookie gespeichert. Dieses wird mit jedem Request an den Server übermittelt. So kann der Server davon ausgehen, dass es sich beim Client um den Inhaber der Session ID handelt.

4.7.2 Token-Based-Authentication

Ein aktuellerer Ansatz, ist es, einen signierten Token im Header jedes Requests der zum Server gesendet wird anzuhängen. Diese Methode bietet folgende Vorteile:

Cross-Domain / CORS Cross-Domain-Cookies und Cross-Domain-Requests funktionieren aufgrund von Sicherheitsrichtlinien der Browser nicht. Ein Token-basierter Ansatz erlaubt es XHR Aufrufe an jeden Server mit jeder Domain zu veranlassen, weil die Informationen im HTTP Header enthalten sind.

Stateless (Server-Side-Scalability) Es wird kein Session-Store benötigt. Der Token enthält alle Benutzerinformationen der Rest des "States" liegt im Local Storage auf der Client-Seite.

Decoupling Man ist nicht an ein Authentifizierungs-Schema gebunden. Der Token kann überall generiert werden, sprich die API kann von überall her aufgerufen werden.

Mobile-Ready Cookies sind in nativen mobilen Apps/APIs meist schlecht unterstützt.

CSRF Cross-Site Requests stellen kein Problem dar. Es gibt kein Authentifizierungs-Cookie, das wiederverwendet werden könnte.

Standard Es gibt bereits einen Standard. (JSON Web Token, JWT)

Die beiden Methoden werden in **??** grafisch dargestellt.

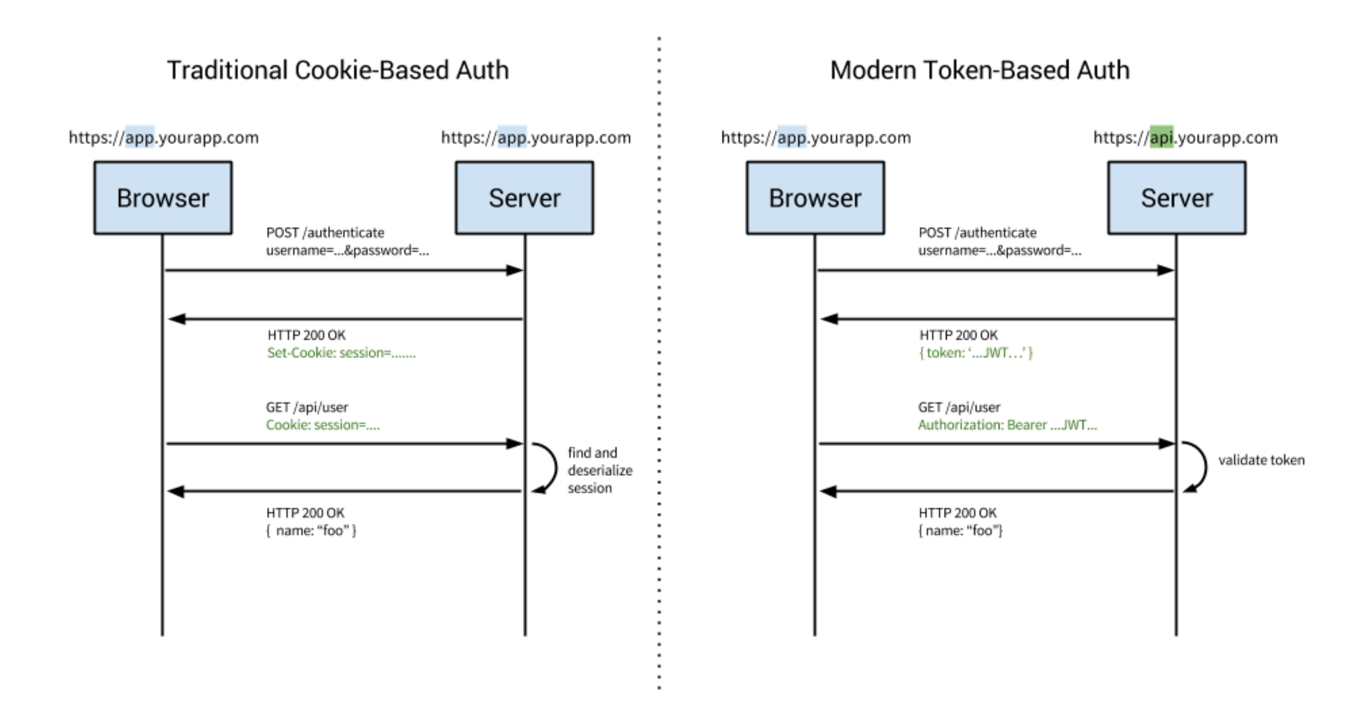


Abbildung 4.5: Cookie vs. Token Auth

4.7.3 Fazit

Entscheid 4.7: Session-Authentifizierung

Da wir die API vom Frontend so weit wie möglich trennen wollen, setzen wir auf die modernere Variante mittels JWT. So könnte die API auch aus einem Client mit anderer Domain verwendet werden.

5 Design und Implementation

5.1 Architektur

Daten-Lieferanten liefern ihre Daten per Web-Interface oder REST-API. Diese Daten werden in der Datenbank gespeichert und im Web-Interface angezeigt. Daten-Nutzer rufen diese wieder über das Web-Interface oder per REST-API ab.

Dritt-Entwickler können eigene Komponenten entwickeln und beisteuern, wie in **??** zu sehen ist:

Parser nehmen Daten entgegen und bereiten sie zur Weiterverarbeitung auf

Formatter formatieren Daten in das vom Benutzer gewünschte Format

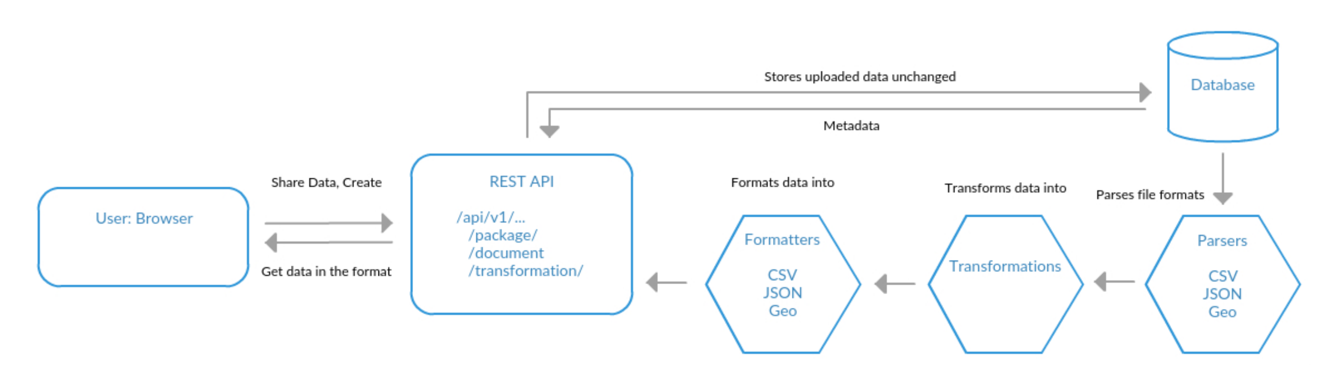


Abbildung 5.1: Grobe Architektur-Übersicht

5.1.1 Komponenten und Packages

?? erläutert die Package-Struktur und Inhalt der Python Module. Alle Python Module die sich im `plugins` Modul befinden, werden automatisch geladen. Darin befinden sich auch Beispiel-Plugins einer ODHQL-Funktion sowie zur Erweiterung der Format-Unterstützung.

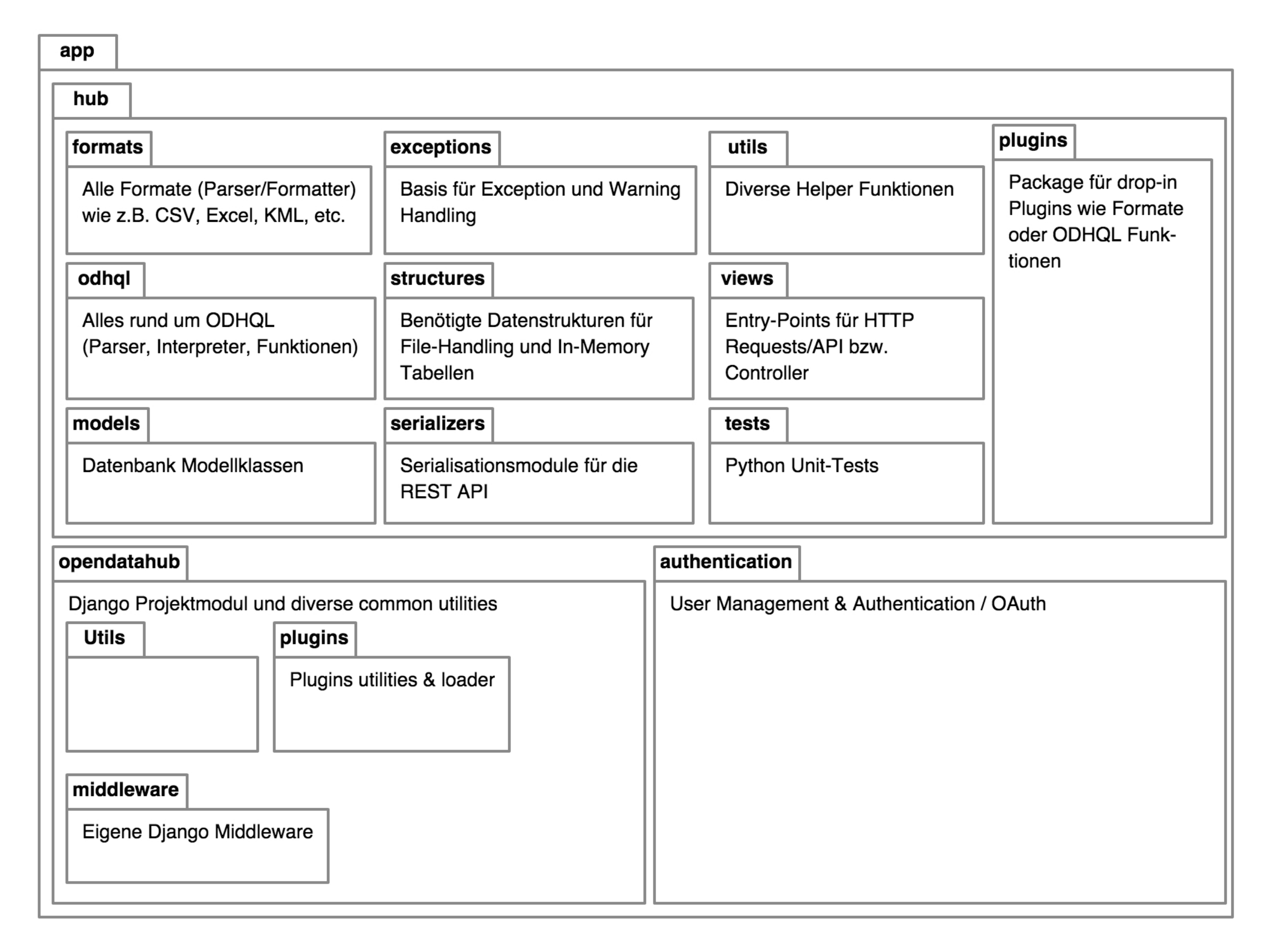


Abbildung 5.2: Python Packages

5.1.2 Klassen

Im folgenden werden einige besonders wichtige Klassen dargestellt.
?? zeigt die erstellten Django-Models[1]. Diese sind alle von `django.db.models.base.Model` abgeleitet.

[1] siehe auch https://docs.djangoproject.com/en/1.8/topics/db/models/

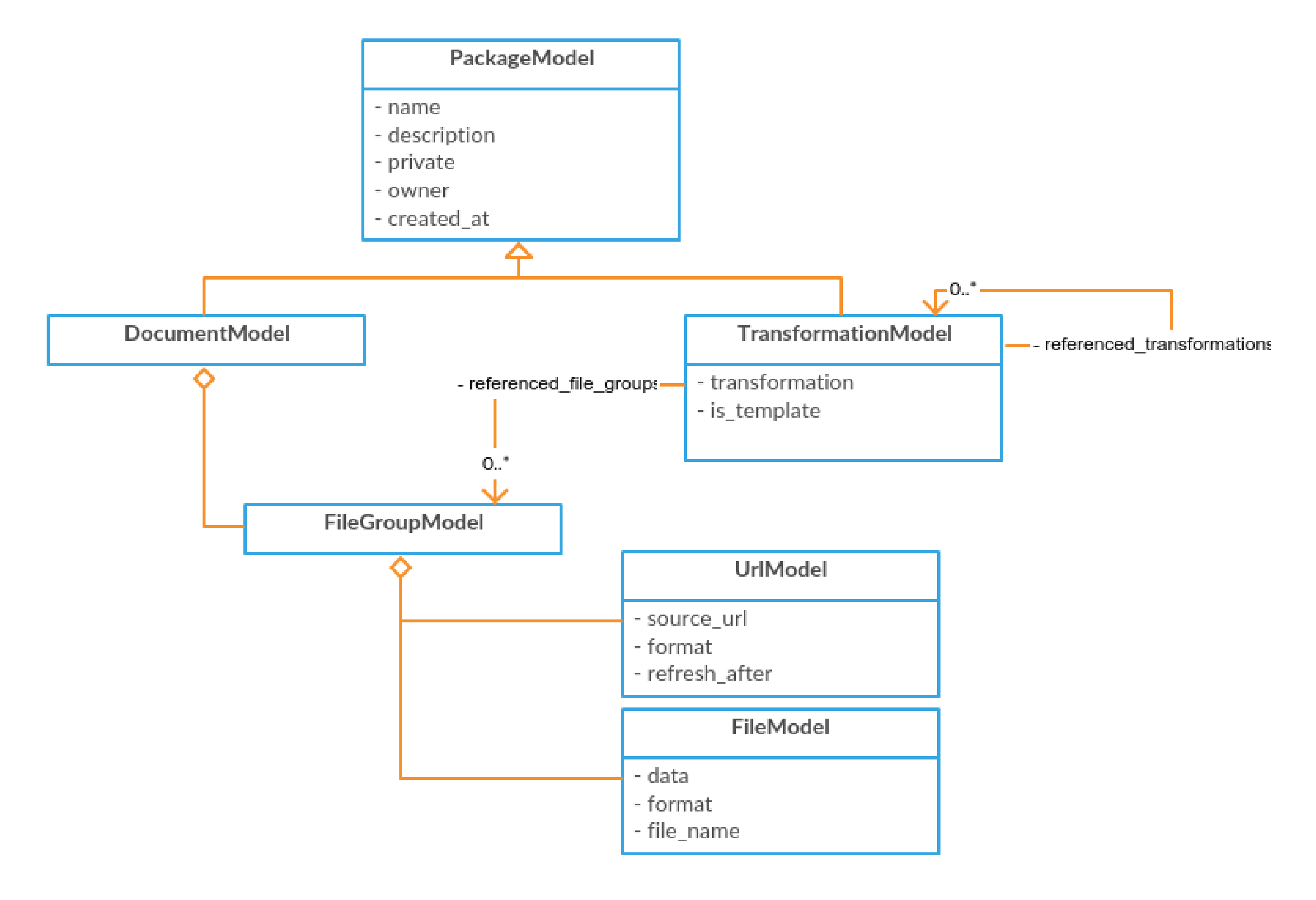


Abbildung 5.3: Django Models

5.2 Transformationssprache ODHQL

In diesem Kapitel wird die Transformationssprache OpenDataHub Query Language (ODHQL) erläutert.

5.2.1 Syntax

Die Syntax von ODHQL orientiert sich stark an SELECT aus ANSI SQL.
Eine formale Beschreibung der Transformationssprache ist in **??** zu finden.

5.2.2 Unterstützte Features

?? beschreibt die unterstützten Features:

Anforderung	Beschreibung
Mapping	Felder können mit oder ohne Alias angegeben werden. Falls kein Alias vorhanden ist, wird der Feldname beibehalten. Es ist möglich, Sonder- und Leerzeichen zu verwenden, indem der Feldname in doppelten Anführungszeichen geschrieben wird.
Default-Werte	Unterstützt werden Integer, Float, sowie Strings (in einfachen Anführungszeichen).
Joins	Es muss immer mindestens eine Datenquelle angegeben werden. Falls mehr verwendet werden sollen, muss mindestens eine Join-Bedingung vorhanden sein (es werden mehrere Join-Bedingungen unterstützt, jedoch nur per "and"-Verknüpfung). Neben Inner Joins werden auch Left, Right und Full Outer Joins unterstützt.
Filter/Prädikate	Vorhandene Filter beinhalten: "is null", "in (...)", relationale Operatoren wie "=", "<", etc., Prädikate und "like 'regex' " (letzteres ist eine Abweichung von ANSI SQL, welches eine separate Syntax für Like-Filter hat)
Erweiterbare Funktionen	Alle Funktionen werden über eine Function Registry in Python aufgerufen und können daher einfach erweitert werden.
Sortierung	Es kann eine Order By-Klausel angegeben werden. Falls Unions verwendet werden, muss die Order By-Klausel am Ende angegeben werden, nicht pro Query.
Unions	Mehrere Queries können per Union zusammengefügt werden. Die Implementation entspricht dem UNION ALL aus SQL, es findet also keine Deduplizierung statt.

Tabelle 5.2: Durch ODHQL implementierte Features

Jede Transformation definiert eine Liste von Feldern oder Werten, sowie mindestens eine Datenquelle.

5.2.3 Beispiele

Folgende Elemente sind in der Feld-Liste erlaubt:

Feld Name des Feldes, mit dem Namen oder Alias der Datenquelle als Präfix. Feldnamen und Aliase können in doppelten Anführungszeichen geschrieben werden, z.B. wenn Sonder- oder Leerzeichen verwendet werden sollen. Zu beachten ist, dass Feldnamen immer einen Präfix brauchen, auch ausserhalb der Feld-Liste.

```
a.beschreibung, a.geom as geometry, a."Zentroid X [m]"
```

Wert Integer, Float, Boolean, Null oder String in einfachen Anführungszeichen. Es muss zwingend ein Alias angegeben werden.

```
1 as one, 3.14 as pi, false as active, 'sun' as local_star
```

Funktion Ein Funktionsaufruf besteht aus einem Namen sowie einer Liste von Argumenten in runden Klammern. Funktionsaufrufe können verschachtelt werden.

```
cast(a.pi, 'float') as pi,
concat('POINT(', cast(v0.x, 'string'), ' ', cast(v0.y, 'string'), ')')) as geometry
```

Datenquellen können per Join verknüpft werden.

```
from "Gebäude" as geb join Strassen as str on geb.str_id = str.id
```

Unterstützt werden Inner Join, Left, Right und Full Outer Join. Dabei wird jeweils die einfachste Syntax[2] verwendet.
Als Filter können folgende Ausdrücke verwendet werden:

Relationale Operatoren Unterstützt werden die Operatoren '=', '!=', '<', '>', '<=', '>='.

```
a.nr > 4, a.active = false
```

Like Anders als bei SQL verwendet der Like-Operator von ODHQL implementationsbedingt Reguläre Ausdrücke (Python Syntax[3]).

```
a.beschreibung like 'Z[uü]rich', a.beschreibung not like 'Z[uü]rich'
```

In Sucht den Ausdruck in einer Liste von Elementen.

```
a.nr in (3, 4, 5), a.nr not in (2, 6, 7)
```

Null-Check Prüft, ob ein Ausdruck Null (bzw. None) ist.

```
a.optional_field is null, a.mandatory_field is not null
```

Die Resultate mehrerer Queries werden mit Union zusammengehängt. Dies verhält sich wie "union all" in SQL, d.h. es findet keine Deduplizierung der Daten statt.

[2] `join`, `left join`, `right join` und `full join`.
[3] Siehe https://docs.python.org/2/library/re.html#regular-expression-syntax

```
select a.description, a.geometry from A as a
 union
select b.description, b.geometry from B as b
```

Zu beachten ist, dass die Feld-Listen der Queries kompatibel sein müssen.

5.2.4 Abstrakter Syntax-Baum (AST)

Der ODHQL-Parser analysiert die Abfrage in Text-Form und wandelt diese in einen Abstrakten Syntax Baum (AST) um. Dieser wird anschliessend vom Interpreter verwendet, um die Transformation auf die Daten anzuwenden.

?? zeigt die Toplevel-Struktur des Abstrakten Syntax-Baums.

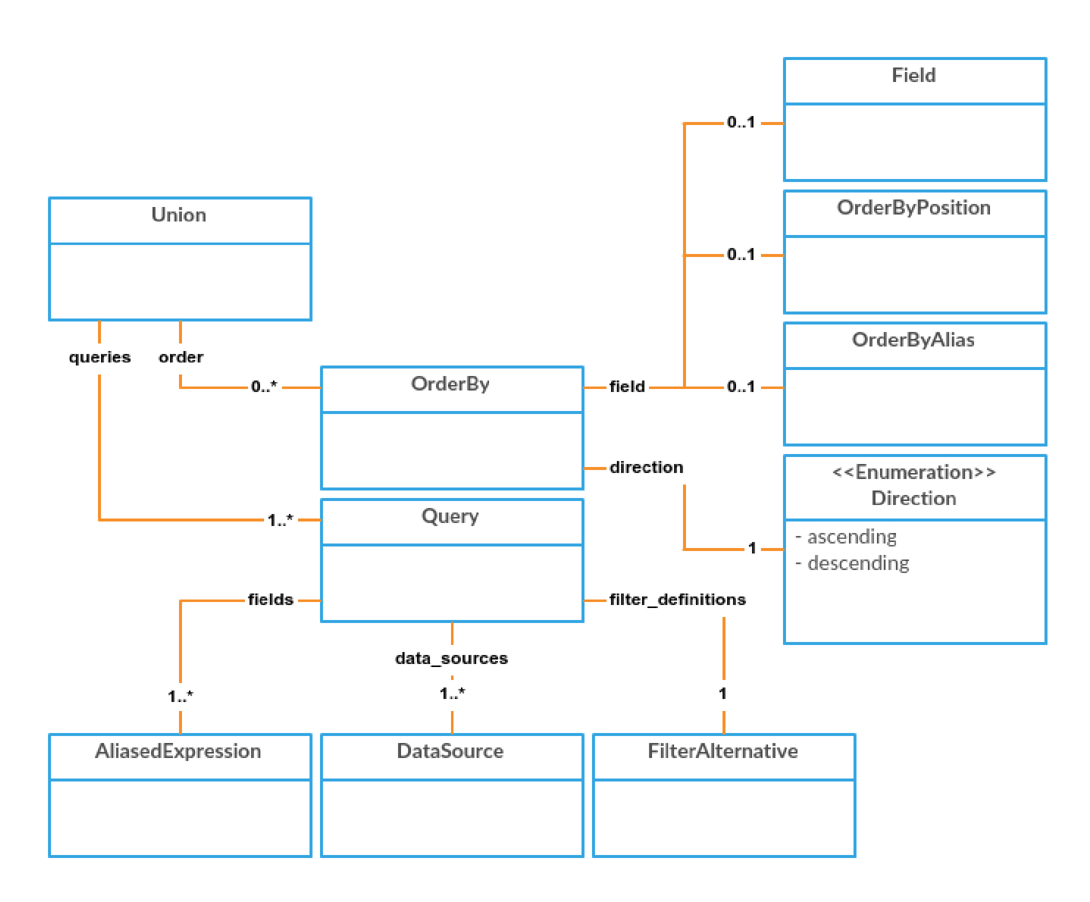


Abbildung 5.4: AST: Unions

Die in **??** beschriebenen Ausdrücke repräsentieren Felder, Werte, Funktionsaufrufe, etc.

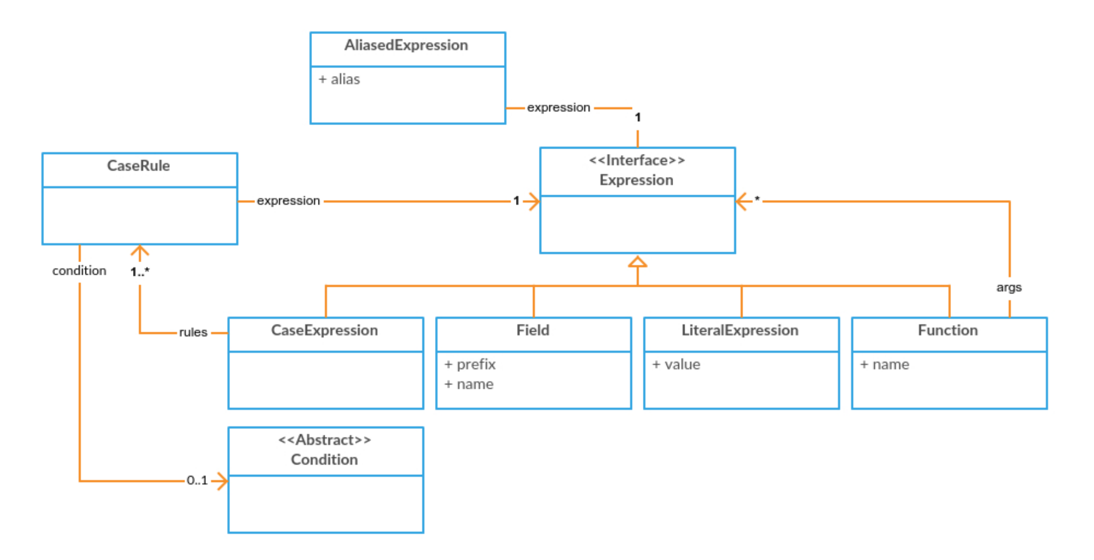


Abbildung 5.5: AST: Ausdrücke

?? zeigt auf, wie Datenquellen repräsentiert werden.

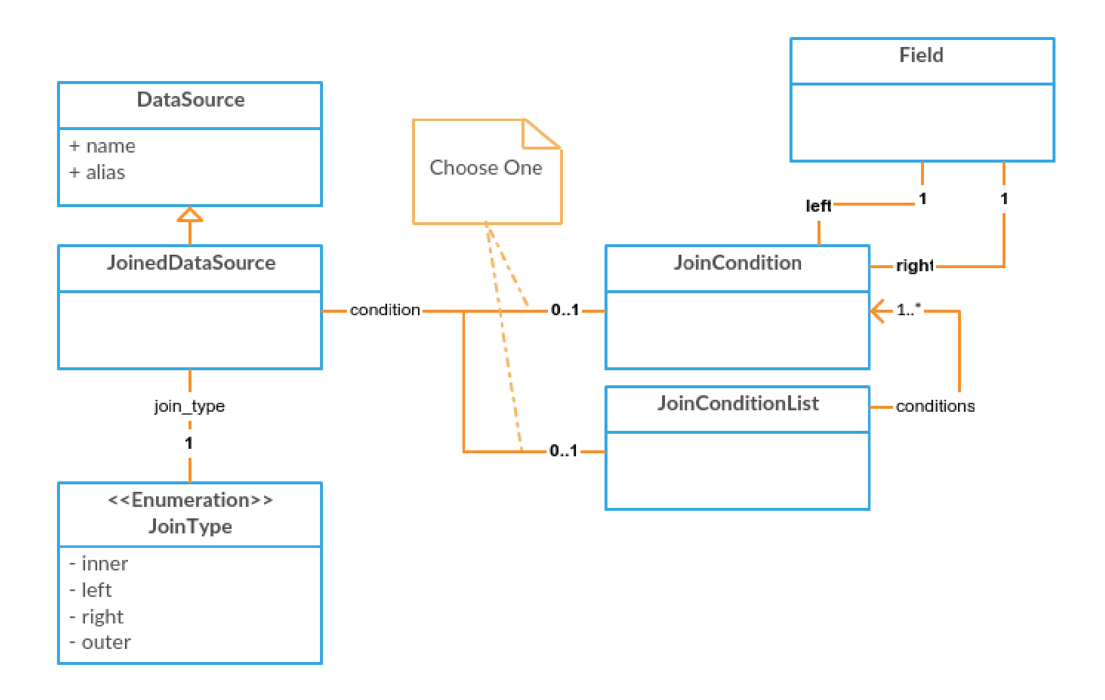


Abbildung 5.6: AST: Datenquellen

Filter werden wie in **??** beschrieben dargestellt.

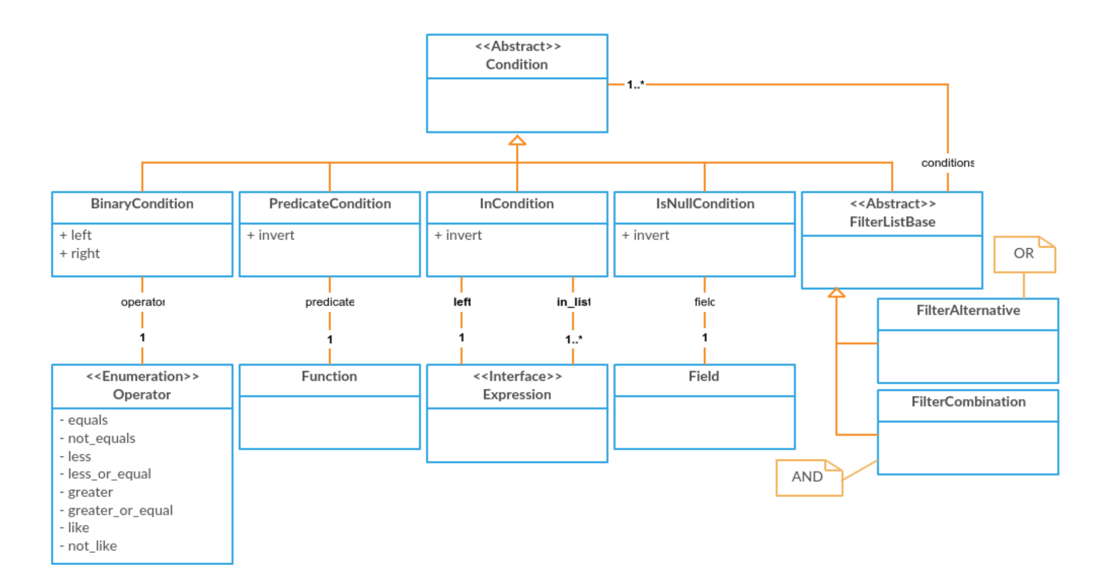


Abbildung 5.7: AST: Filter

5.2.5 Interpreter

Mit Pandas und NumPy ist bereits eine API für viele SQL-ähnliche Operationen wie Selektion, Joins und diverse String- und Geometriefunktionen (GeoPandas Erweiterung) vorhanden. Die Herausforderung bei der Entwicklung des Interpreters liegt somit einerseits in einer performanten und möglichst fehlerfreien Ausführung, andererseits der Erweiterbarkeit durch eigene Funktionen.

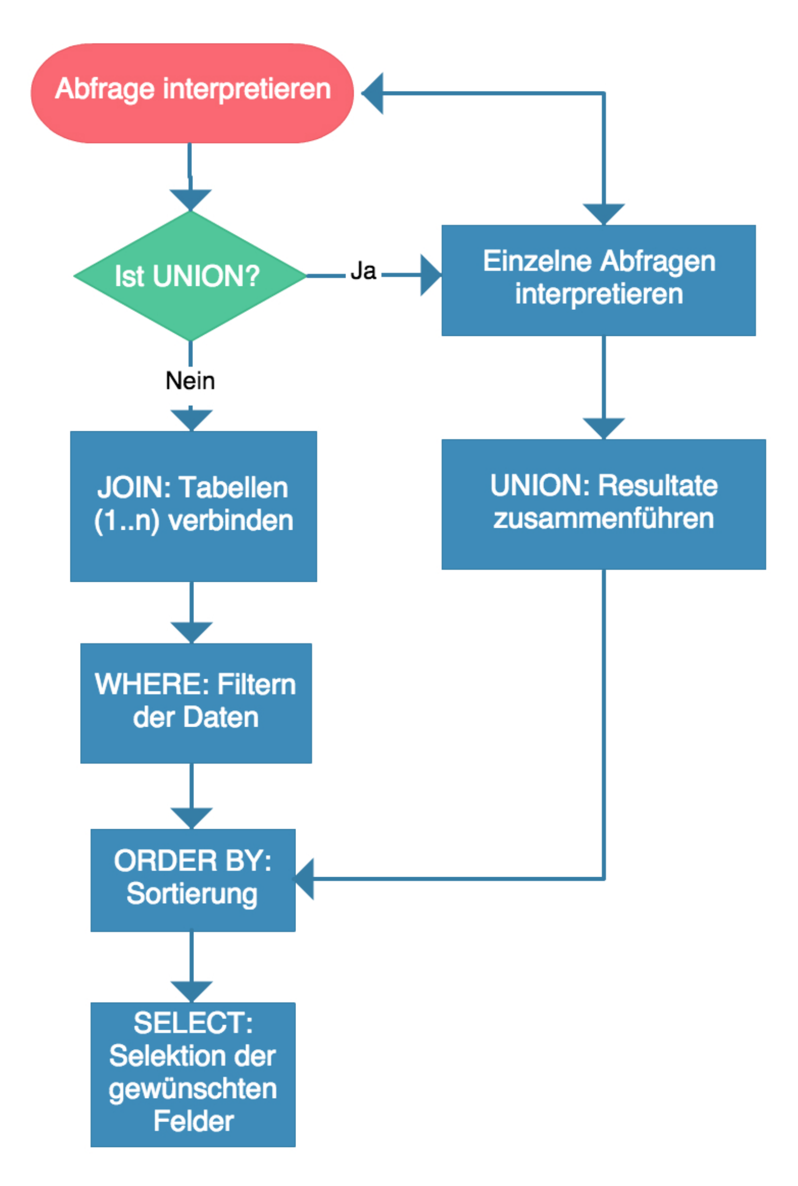


Abbildung 5.8: High-level Ablauf der Abfrage-Interpretation

5.2.6 Funktionen

Die Funktionen der ODHQL wurden möglichst erweiterbar implementiert. Neue Funktionen erben von der Klasse `VectorizedFunction` und implementieren die Methode `apply`.
Durch die Vererbung und der zugrunde liegenden Python-Metaklasse wird die neue Funktion automatisch registriert und ist sofort mit dem Klassennamen verfügbar. Der Name kann durch das statische Attribut `name` auch explizit gesetzt werden. Die Anzahl der Funktionsparameter werden vom Interpreter bzw. von der Execution Engine der Funktionen via Reflection sichergestellt. Für die Überprüfung der Argumente ist jede Funktion selbst zuständig. Die Parent-Klasse bietet jedoch bereits einige Assertion-Methoden z.B. für Datentypen, Listen und Geometrien.
Ein Beispiel einer solchen Funktion ist in **??** ersichtlich. Hierbei wird nach der Überprüfung der Argumente direkt an die Pandas-Methode `str.pad` weiterdelegiert.

Programmcode 5.1: Beispiel-Implementation der ODHQL-Funktion `PAD()`

```
from hub.odhql.functions.core import VectorizedFunction

class Pad(VectorizedFunction):
    name = 'PAD'

    def apply(self, strings, width, side):
        self.assert_str('string', strings)
        self.assert_in('side', side, ['left', 'right', 'both'])
        self.assert_int('width', width)
        return strings.str.pad(width, side)
```

5.3 Caching

Aufgrund der Entscheidung alle Daten verlustfrei im Originalformat zu speichern (**??**), sowie der Verwendung von In-Memory Tabellen-artigen Datenstrukturen, muss sichergestellt werden, dass nicht bei jeder Operation die Originaldaten von Format-Parser neu eingelesen werden müssen.

Aus diesem Grund wurde ein dreistufiges Caching-Konzept für die gesamte Applikation implementiert. Es werden die Caching-Mechanismen von Django mit einigen Modifikationen/Erweiterungen verwendet. Folglich können die Caching-Backends jederzeit durch beliebige von Django unterstützte Backends wie z.B. Memcached, Filesystem, etc. ersetzt werden.

5.3.1 Caches

?? zeigt eine Übersicht der in ODH eingesetzten Caches und deren Storage-Backends.

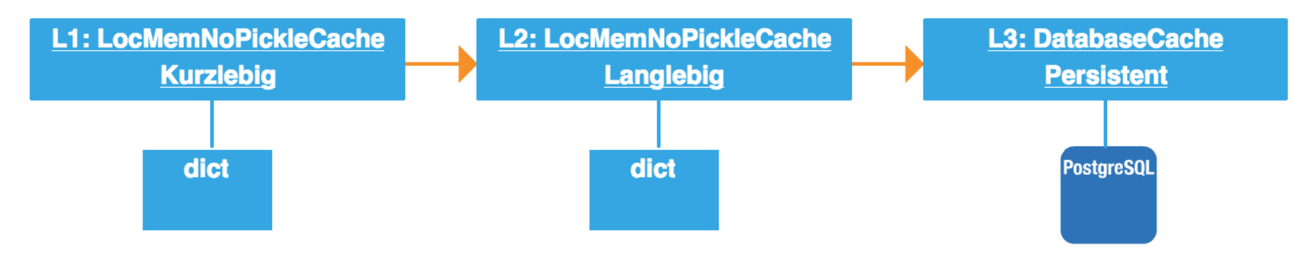


Abbildung 5.9: Übersicht der Caches

Cache	Beschreibung
Level 1	Der Level 1 Cache ist ein In-Memory Cache und sehr kurzlebig (60 s). Dieser wird hauptsächlich verwendet, um Zwischenresultate wie Format-Konversionen oder Resultate einer Abfrage zu speichern, damit der Benutzer im Webfrontend die Daten sofort mittels Pagination anschauen kann.
Level 2	Der Level 2 Cache ist ebenfalls ein In-Memory Cache, allerdings werden die Daten länger vorgehalten (300 s). Darin werden hauptsächlich Daten aus dem Level 3 Cache für mehrere Minuten zwischengespeichert, um Abfrage- und Deserialisierungs-Overhead zu vermeiden.
Level 3	Der Level 3 Cache ist mit Hilfe eines Datenbank-Backends implementiert. Der Cache speichert eingelesene Dateien im intermediären Format (d.h. DataFrame Objekte) sowie gespeicherte Transformationen ohne Online-Datenquellen.

Tabelle 5.4: Beschreibung der Caches

5.3.2 Invalidierung

Überall wo Caching-Mechanismen eingesetzt werden, stellt sich das grosse Problem der Cache-Invalidierung. Wann müssen welche Cache-Einträge invalidiert oder gelöscht werden um falsche oder veraltete Resultate zu verhindern. Da bei ODH jeweils die DataFrame Objekte von Dateien und Transformationen gespeichert werden, muss wie folgt invalidiert werden:

- sobald die Daten einer Online-Quelle[4] sich ändern bzw. neu Angefordert wurden,
- wenn eine Transformation modifiziert wird, welche von anderen Transformationen verwendet wird (rekursiv),
- oder sobald Daten gelöscht werden.

Hierfür wird eine Zwischentabelle in der Datenbank benötigt, um festzuhalten, welche Transformation/Daten von welchen Transformationen referenziert werden. Diese Information wird dann zugleich für die Navigation zwischen Daten und Transformationen im Frontend wiederverwendet[5]. Um jeweils alle zu einem Cache-Key[6] gehörenden Einträge zu löschen, wurden die eingebauten Django-Caches so erweitert, dass für alle verwendeten Caching-Provider[7] ein kaskadierendes Löschen möglich ist. So lassen sich beispielsweise durch das Löschen von `('TRF',12)` alle gespeicherten und konvertierten Daten, die zu der Transformation mit der ID 12 gehören, "kaskadierend" löschen[8]

[4] Beispielsweise ein WFS Webservice
[5] Welche Daten/Transformationen haben assoziierte Transformationen
[6] Ursprünglich ein Tupel, z.B. `('TRF',10,'CSV')`
[7] In-Memory und Datenbank
[8] Also auch `('TRF',12,'KML')` etc.

5.4 Format-Unterstützung

5.4.1 Unterstützte Formate

?? beschreibt die implementierten Formate.

Format	Implementiert via
CSV (inkl. GeoCSV)	(Geo)Pandas
Excel (xls, xlsx)	Pandas
GML	ogr2ogr
GeoJSON	GeoPandas
GeoPackage[9]	GeoPandas
Interlis 1	ogr2ogr
Interlis Modell (ili)	custom
JSON	Pandas
KML	fastkml, custom
ESRI Shapefile	GeoPandas
WFS (nur Parser)	ogr2ogr
XML	custom

Tabelle 5.6: Beschreibung der implementierten Formate

GeoPandas verwendet (via Fiona[10]) ebenfalls GDAL.

5.4.2 Probleme mit GDAL

Für einen grossen Teil der Formate wird GDAL verwendet, entweder via GeoPandas/Fiona oder direkt per ogr2ogr.
Folgende GDAL-Versionen stehen zum aktuellen Zeitpunkt zur Verfügung:

- 1.10.1
- 1.11.0 - 1.11.2
- 2.0.0beta1

Die Entwicklungsumgebung besteht aus einer virtuellen Maschine mit Ubuntu 14.04 LTS, für welches ein GDAL-Package in Version 1.10.1 vorhanden ist. Auf Wunsch der Betreuers, wenn möglich Unterstützung für das relativ neue Format GeoPackage einzubauen, wurden diversen Möglichkeiten geprüft. GDAL 1.10.1 unterstützt GeoPackage nicht. Im Versuch, diese Unterstützung einzubauen wurde GDAL selbst kompiliert. Dem Betreuer waren diverse Probleme mit GDAL 1.11.2 bekannt, weswegen 1.11.1 gewählt wurde.
In Tests wurden folgende Probleme gefunden:

[9] Deaktiviert, siehe **??**
[10] https://pypi.python.org/pypi/Fiona

- In GDAL 1.11.0 bis 2.0.0beta1 führt der Versuch, nach Interlis 1 zu konvertieren, zu einem Segmentation Fault (segfault). Das Problem wurde vom Betreuer an den Treiber-Entwickler weiter gemeldet. In aktuellen nightly-Versionen soll dies behoben sein - dies wurde nicht verifiziert.
- Der GeoPackage-Treiber wurde in GDAL 1.11.0 hinzugefügt. Leider kann er in dieser Version nur als unausgereift bezeichnet werden:
  - Layer-Namen werden ohne Bereinigung oder Maskierung als Tabellen-Namen verwendet, was zu Exceptions seitens SQLite führt.
  - Wenn die Ursprungs-Daten bereits eine Spalte namens "fid" beinhalten, bricht die Konvertierung ab wegen doppelter Spalten-Namen, da der Treiber diese Spalte selbst nochmals hinzu generiert.
  - Reine Daten-Tabellen werden nicht unterstützt. Diese sind zwar nicht Bestandteil des reinen GeoPackage-Standards, können aber mit Extended GeoPackage realisiert werden. Dies ist in GDAL erst ab Version 2.0.0 möglich.

GeoPackage-Support wurde in GDAL 2.0.0 massiv verbessert. Diese Version ist bei Abschluss der Arbeit jedoch noch in der Beta-Phase, weshalb diverse unserer Abhängigkeiten dazu noch nicht kompatibel sind.

Entscheid 5.1: GDAL-Version

Zum Abgabe-Zeitpunkt sind folgende GDAL-Versionen verfügbar: 1.10.1, 1.11.0 - .2 und 2.0.0beta1.
Wegen Problemen mit mehreren Treibern entfallen die Versionen 1.11.x.
Version 2.0.0beta1 entfällt wegen mangelnder Unterstützung in unseren Abhängigkeiten.
Daher wird vorerst GDAL 1.10.1 eingesetzt.

5.4.3 Interlis 2

Entscheid 5.2: Kein Interlis 2 Support

Aufgrund von Problemen mit dem Format-Support in ogr2ogr wurde in Absprache mit dem Betreuer entschieden, Unterstützung für Interlis 2 nicht zu implementieren. Siehe dazu das Sitzungsprotokoll vom 15.04.2015.

5.4.4 Erweiterung

Die Unterstützung für neue Formate kann relativ einfach hinzugefügt werden. Dazu müssen Subklassen für folgende Python-Klassen erstellt werden:

hub.formats.core.Format Dient zur Identifikation des Formats von Datenquellen (z.B. Dateien)

hub.formats.core.Formatter Erhält DataFrames und produziert Dateien, welche der Benutzer herunterladen kann.

hub.formats.core.Parser Erhält Daten (aus Dateien, WFS, ...) und produziert DataFrames, welche von ODH weiterverwendet werden können.

Für die Implementation via ogr2ogr stehen im Modul hub.formats.geobase Hilfsklassen zur Verfügung. Durch Vererbung und die verwendete Python-Metaklasse wird das neue Format automatisch registriert und kann sofort verwendet werden.
Ein Beispiel ist in **??** zu sehen. Darin wird die Unterstützung für ESRI Shapefile implementiert. Der Formatter verwendet hierzu die Hilfsklasse GeoFormatterBase, der Parser jedoch direkt GeoPandas.

Programmcode 5.2: Beispiel-Implementation der Format-Unterstützung für ESRI Shapefile

```
import geopandas as gp
import os

from hub.formats import Format, Parser
from hub.formats.geobase import GeoFormatterBase
from hub.utils import ogr2ogr

if 'ESRI Shapefile' in ogr2ogr.SUPPORTED_DRIVERS:
    class Shapefile(Format):
        label = 'ESRI Shapefile'
        ogr_format = ogr2ogr.SHP

        description = """
        Ein ursprünglich für die Firma ESRI entwickeltes Format für Geodaten.
        """

        extension = 'shp'

        @classmethod
        def is_format(cls, file, *args, **kwargs):
            return file.extension == 'shp'

    class ShapefileFormatter(GeoFormatterBase):
        targets = Shapefile,

        supported_types = {'Point', 'LineString', 'LinearRing', 'Polygon',
         ↪  'MultiPoint'}

        @classmethod
        def format(cls, dfs, name, format, *args, **kwargs):
            return super(ShapefileFormatter, cls).format(dfs, name, format, 'ESRI
             ↪  Shapefile', 'shp', *args, **kwargs)

    class ShapefileParser(Parser):
        accepts = Shapefile,

        @classmethod
        def parse(cls, file, format, *args, **kwargs):
            with file.file_group.on_filesystem() as temp_dir:
```

```
            return gp.read_file(os.path.join(temp_dir, file.name), driver='ESRI
             ↪   Shapefile')
```

5.5 API

Die Webapplikation kommuniziert mit dem Server über ein Representational State Transfer (REST) API.

5.5.1 Konzepte

Folgende Konzepte werden von mehreren Endpunkten verwendet und werden daher separat aufgelistet.

Paging

Paging ermöglicht das limitieren der Anzahl Resultate, welche eine Abfrage liefert. Die Einträge pro Seite werden mit dem Parameter `count` gesteuert, die Seite mit `page`.

Suche

Für die Suche werden folgende Parameter verwendet:

Parameter	Beschreibung
filter[name]	Filtert anhand des Namens
filter[description]	Filtert anhand der Beschreibung
filter[search]	Filtert sowohl mit Namen als auch mit der Beschreibung
filter[mineOnly]	Liefert nur Dokumente, die dem eingeloggten Benutzer zugeordnet sind

Tabelle 5.8: Such-Parameter

Sortierung

Sortiert die Resultate der Anfrage. Dazu kann der Parameter `sorting[key]=asc|desc` verwendet werden, mit folgenden Werten als `key`: `name`, `description`, `private`, `owner` und `created_at`.

Preview Liefert eine Vorschau der Daten. Enthalten ist der Unique Name, welcher in Abfragen verwendet werden kann, eine Liste der Felder und deren Typen sowie ein paar Daten-Zeilen. Die Anzahl der gelieferten Daten-Zeilen kann per Paging gesteuert werden (siehe **??**).

Programmcode 5.3: Preview für ein Dokument

```
[
  {
    "type":"preview",
```

```
      "unique_name":"ODH4_Bahnhoefe",
      "parent":"4",
      "types":{
         "TYPE":"BIGINT",
         "fid":"TEXT",
         "CNTRYNAME":"TEXT",
         "DUP_NAME":"TEXT",
         "PROV1NAME":"TEXT",
         "LEVEL":"BIGINT",
         "geometry":"GEOMETRY",
         "NATION":"BIGINT",
         "CONURB":"TEXT",
         "name":"TEXT"
      },
      "url":"http://beta.opendatahub.ch/api/v1/document/4/preview/",
      "columns":[ "TYPE", "fid", "geometry", "DUP_NAME", "PROV1NAME", "LEVEL",
       ↪   "CNTRYNAME", "NATION", "CONURB", "name" ],
      "data":[
         {
            "TYPE":"3",
            "fid":"F0",
            "CNTRYNAME":"Switzerland",
            "DUP_NAME":"N",
            "PROV1NAME":"Tessin",
            "LEVEL":"10",
            "geometry":"POINT (45.90247783251107 76645.7742365048)",
            "NATION":"41",
            "CONURB":"NULL",
            "name":"Chiasso Rail"
         },
         ...
      ],
      "count":1781,
      "name":"Bahnhoefe"
   }
]
```

Download

Fordert einen Download der Daten an. Das gewünschte Format wird mit dem Parameter `fmt` angegeben. Gültige Werte sind die von **??** gelieferten Namen. Ohne Format-Angabe werden die Original-Daten geliefert.
Hinweis: Dokumente können nicht als ganzes heruntergeladen werden – stattdessen müssen die File Groups für diesen Zweck verwendet werden.

5.5.2 Root - GET /api/v1/

Als Prefix für die API URLs wird /api/v1/ verwendet, was eine spätere versionierte Erweiterung erlaubt.

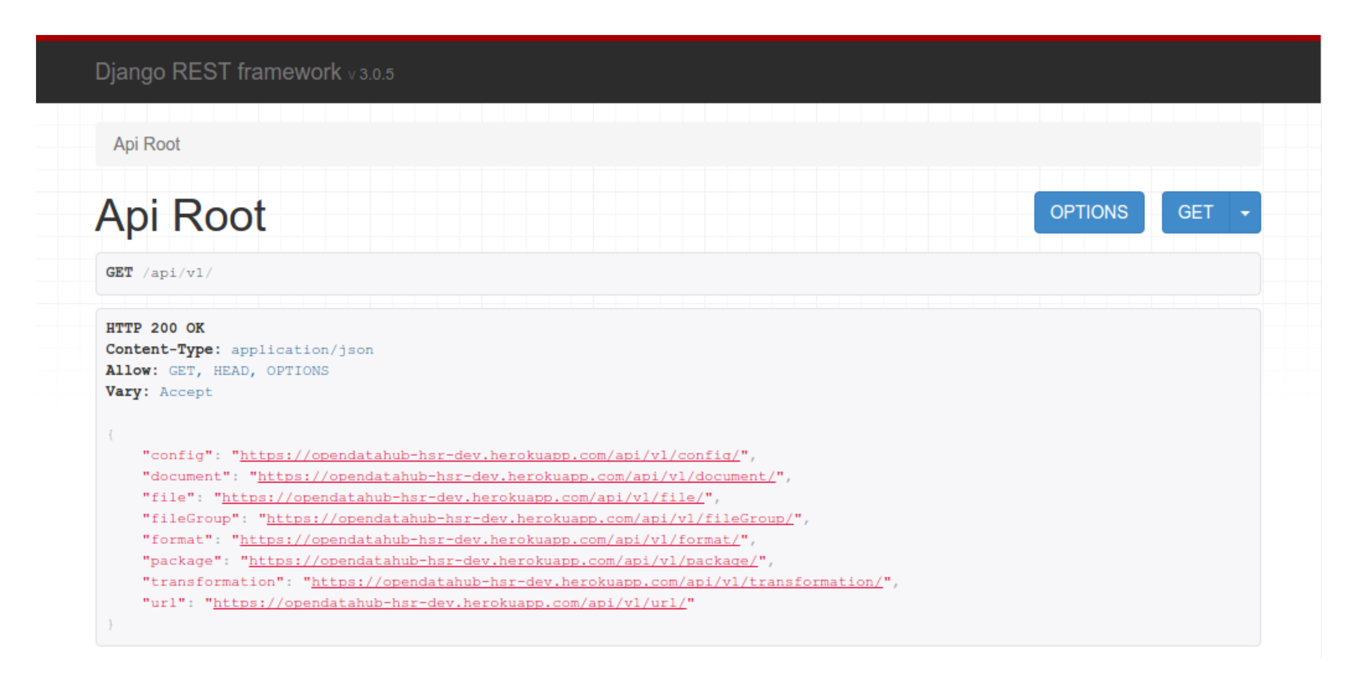


Abbildung 5.10: API: Übersicht

Da für die Implementation des API die Django-Erweiterung `rest_framework`[11] verwendet wurde, kann die API gut auch manuell ausprobiert werden. Dabei ist zu beachten, dass die Schrägstriche am Ende der URLs nicht optional sind.

5.5.3 Konfiguration - GET /api/v1/config/

Gibt die für den HTML5 Client notwendigen Konfigurationsinformationen zurück.

Programmcode 5.4: Beispielantwort für GET /api/v1/config/

```
{
  "GITHUB_PUBLIC": "8ef558ed3fb0f5385da5",
  "FACEBOOK_PUBLIC": "401520096685508",
  "PACKAGE_PREFIX": "ODH",
  "TRANSFORMATION_PREFIX": "TRF"
}
```

5.5.4 Liste der unterstützten Formate - GET /api/v1/format/

Liefert die Liste der unterstützten Formate. Neue Formate erscheinen automatisch, sobald sie registriert sind, vorausgesetzt es existiert ein Formatter für das Format.

[11] http://www.django-rest-framework.org/

Programmcode 5.5: Beispielantwort für GET /api/v1/format/

```
[
    {
        "name": "JSON",
        "label": "JSON",
        "description": "JavaScript Objekt-Notation. Nützlich zur Wiederverwendung in
         ↪  Webapplikationen.",
        "example": "",
        "extension": "json"
    },
    {
        "name": "GML",
        "label": "GML",
        "description": "Geometry Markup Language (GML) ist ein Dateiformat zum
         ↪  Austausch von Geodaten.",
        "example": "",
        "extension": "gml"
    },
    ...
]
```

5.5.5 Packages

"Package" wird als Überbegriff für Dokumente, welche von Nutzern bereitgestellt wurden oder für Transformationen verwendet. Packages haben Attribute wie "Name" oder "Beschreibung", können eine Vorschau erzeugen und der Benutzer kann einen Download anfordern.

Package-Liste - GET /api/v1/package/

Liefert eine Liste der Packages.

Programmcode 5.6: Beispielantwort für GET /api/v1/package/

```
{
    "count": 37,
    "next": "http://beta.opendatahub.ch/api/v1/package/?page=2",
    "previous": null,
    "results": [
        {
            "id": 1,
            "url": "http://beta.opendatahub.ch/api/v1/package/1/",
            "name": "Test mockaroo.com.csv",
            "description": "Testdaten (Originalformat: CSV)",
            "private": false,
            "owner": {
                "id": 1,
                "username": "testuser",
                "first_name": "",
                "last_name": ""
```

```
            },
            "created_at": "2015-06-01T12:35:31.480135Z",
            "type": "document",
            "preview": "http://beta.opendatahub.ch/api/v1/document/1/preview/",
            "template": false
        },
        {
            "id": 2001,
            "url": "http://beta.opendatahub.ch/api/v1/package/2001/",
            "name": "TROBDB: Baustellen Februar 2015",
            "description": "TROBDB: Baustellen Februar 2015",
            "private": false,
            "owner": {
                "id": 1,
                "username": "testuser",
                "first_name": "",
                "last_name": ""
            },
            "created_at": "2015-06-01T12:35:45.794037Z",
            "type": "transformation",
            "preview":
             ↪  "http://beta.opendatahub.ch/api/v1/transformation/2001/preview/",
            "template": false
        },
        ...
    ]
}
```

Die Endpunkte (**??**), Suche (**??**) und Sortierung (**??**) unterstützten Paging.

Package-Details - GET /api/v1/package/<id>/

Liefert die Details zu einem Package. Dies sind grundsätzlich dieselben Informationen wie in der Package-Liste, jedoch um einige weitere Details ergänzt.
Die Antwort sieht unterschiedlich aus, je nachdem ob das Package ein Dokument (**??**) oder eine Transformation (**??**) ist.

Programmcode 5.7: Beispielantwort für GET /api/v1/package/4/

```
{
    "id": 4,
    "url": "http://beta.opendatahub.ch/api/v1/document/4/",
    "name": "Test Bahnhoefe.gml, Bahnhoefe.gfs, Bahnhoefe.xsd",
    "description": "Testdaten (Originalformat: GML)",
    "file_groups": "http://beta.opendatahub.ch/api/v1/document/4/filegroup/",
    "private": false,
    "owner": {
        "id": 1,
        "username": "testuser",
        "first_name": "",
        "last_name": ""
```

```
    },
    "created_at": "2015-06-01T12:35:32.241407Z",
    "preview": "http://beta.opendatahub.ch/api/v1/document/4/preview/",
    "type": "document"
}
```


Programmcode 5.8: Beispielantwort für GET /api/v1/package/2001/


```
{
    "id": 2001,
    "url": "http://beta.opendatahub.ch/api/v1/transformation/2001/",
    "name": "TROBDB: Baustellen Februar 2015",
    "description": "TROBDB: Baustellen Februar 2015",
    "transformation": "SELECT t.\"Federführende_Stelle\" as userid,
           substring(t.Bauarbeiten, 1, 100) as title,
           t.Bauarbeiten as description,
           to_char(to_date(concat(extract(t.Dauer, '^((?:\\\\d\\\\d?\\\\.){2})'),
 ↪  nvl(extract(t.Dauer, '(\\\\d{4}) -'), extract(t.Dauer, '(\\\\d{4})$'))),
 ↪  '%d.%m.%Y'), '%Y-%m-%d') as trob_start, -- add year from end date if not present
           to_char(to_date(extract(t.Dauer, '- ([\\\\d\\\\.]+)'), '%d.%m.%Y'),
 ↪  '%Y-%m-%d') as trob_end,
           null as trob_interval,
           'both' as direction,
           null as diversion_advice,
           'CH' as country,
           t.Bauarbeiten as reason,
           t.Strecke as object_name,
           'street' as object_type,
           'closed' as trob_type,
           ST_SetSRID(ST_GeomFromText(CONCAT('POINT(', CAST(t.\"Zentroid X [m]\",
 ↪  'text'), ' ',CAST(t.\"Zentroid Y [m]\", 'text'), ')')), 21781) as geometry
           from \"ODH8_Baustellen Februar 2015\" as t",
    "private": false,
    "owner": {
        "id": 1,
        "username": "testuser",
        "first_name": "",
        "last_name": ""
    },
    "data": "http://beta.opendatahub.ch/api/v1/transformation/2001/data/",
    "is_template": false,
    "preview": "http://beta.opendatahub.ch/api/v1/transformation/2001/preview/",
    "referenced_file_groups":
     ↪  "http://beta.opendatahub.ch/api/v1/transformation/2001/filegroups/",
    "referenced_transformations":
     ↪  "http://beta.opendatahub.ch/api/v1/transformation/2001/transformations/",
    "related_transformations": [
        {
            "id": 2007,
            "name": "TROBDB: Alle Daten"
        }
```

```
    ],
    "type": "transformation"
}
```

File Groups Die `file_group`-URL liefert die Liste der File Groups, die zu diesem Dokument gehören. Siehe auch **??**.

Preview Die Preview-URL liefert eine Vorschau der Daten im Package. Siehe auch **??**.

Referenced File Groups/Transformations Dies sind URLs für File Groups bzw. Transformations, welche in dieser Transformation verwendet wurden.

Template Templates sind abstrakte Transformationen, welche als Vorlage für konkrete Transformationen dienen. Für Templates kann keine Vorschau erstellt werden.

Data URL für den Download der Daten. Siehe **??** für Details.

5.5.6 Dokumente

Dokument-Liste - GET /api/v1/document/

Die Dokument-Liste ist identisch zur Package-Liste, mit der Ausnahme, dass nur Dokumente aufgeführt werden. Siehe **??** für Details.

Dokument-Details - GET /api/v1/document/<id>/

Identisch zu Package-Details, mit der Ausnahme, das nur Dokumente unterstützt werden. Siehe **??** für Details.

Dokument-Erstellung - POST /api/v1/document/

Erstellt ein neues Dokument.
?? führt die erwarteten Parameter auf.

Parameter	Beschreibung
name	Dokument-Name (max. 200 Zeichen).
description	Beschreibung des Dokuments (keine Limitation).
private (default: false)	Steuert ob das Dokument für andere Benutzer sichtbar ist.
format (default: null)	Format der hochgeladenen Daten. Wenn nicht angegeben wird versucht, das Format automatisch zu erkennen.

Tabelle 5.10: Parameter für Dokument-Erstellung

Zusätzlich wird eine der folgenden Optionen benötigt:

file Direkter Datei-Upload. In diesem Fall wird die Anfrage als Multipart Form Request erwartet. Es kann mehr als eine Datei auf einmal hochgeladen werden. Diese werden dann dem selben Dokument zugeordnet, nach Datei-Namen (ohne Erweiterung) in File Groups gruppiert.

url Erstellt ein Dokument mit einer URL als Datenquelle. Dies kann entweder ein WFS-Endpoint oder eine HTTP-URL sein. Zusätzlich kann der Parameter `refresh` angegeben werden, welcher steuert, wie oft die Daten abgerufen werden sollen (in Sekunden). Erlaubt sind Werte von 60 (1 Minute) bis 604800 (1 Woche).

Programmcode 5.9: Beispielanfrage für POST /api/v1/document/

```
{
  "name":"aasdfas",
  "description":"asdfasdf",
  "url":"http://maps.zh.ch/wfs/TbaBaustellenZHWFS",
  "refresh":3600
}
```

Dokument-Update - PATCH /api/v1/document/<id>/

Aktualisiert ein Dokument. Es gelten folgende Einschränkungen:

- Nur die Meta-Daten (Name, Beschreibung, Privat) können aktualisiert werden.
- Nur der Eigentümer eines Dokuments kann aktualisieren.

5.5.7 Transformationen

Transformations-Liste - GET /api/v1/transformation/

Die Transformations-Liste ist identisch zur Package-Liste, mit der Ausnahme, dass nur Transformationen aufgeführt werden. Siehe **??** für Details.

Transformations-Details - GET /api/v1/transformation/<id>/

Identisch zu Package-Details, mit der Ausnahme, das nur Transformationen unterstützt werden. Siehe **??** für Details.

Transformation erstellen - POST /api/v1/transformation/

Erstellt eine neue Transformation.
?? führt die erwarteten Parameter auf.

Parameter	Beschreibung
name	Name der Transformation (max. 200 Zeichen).
description	Beschreibung der Transformation (keine Limitation).
private (default: false)	Steuert ob die Transformation für andere Benutzer sichtbar ist.
transformation	ODHQL Statement, welches die Transformation definiert. Falls die referenzierten Datenquellen nicht existieren wird ein Template erstellt.

Tabelle 5.12: Parameter für Erstellung einer Transformation

Hinweis: Vor der Erstellung einer Transformation sollte geprüft werden, ob das ODHQL Statement korrekt ist. Siehe dazu **??**.

Programmcode 5.10: Beispielanfrage für POST /api/v1/transformation/

```
{
  "name":"Beispiel",
  "description":"Dies ist ein Beispiel für die Dokumentation.",
  "transformation":"SELECT t1.children,\nt1.city,\nt1.country,\nt1.email \nFROM
   ↪  \"ODH2_mockaroo.com\" as t1 \n",
  "private":true
}
```

Transformations-Update - PATCH /api/v1/transformation/<id>/

Aktualisiert eine Transformation. Es können alle in **??** erwähnten Felder aktualisiert werden.

Preview für AdHoc-Transformationen - POST /api/v1/transformation/adhoc/

Erstellt eine Preview für ein ODHQL Statement, ohne die Transformation in der Datenbank zu persistieren. Dies ist nützlich, um eine Vorschau realisieren zu können, während der Benutzer noch am Bearbeiten seines Statements ist.

5.5.8 File Groups

Von Benutzern bereitgestellte Daten werden in sog. File Groups organisiert. Dies sind zusammengehörende Dateien, z.B. die .shx-, .shp- und .dbf-Datei bei einem ESRI Shapefile.

File Group-Liste - GET /api/v1/fileGroups/

Liefert eine Liste aller vorhandene File Groups.

Programmcode 5.11: Beispielantwort für GET /api/v1/fileGroups/

```
[
    {
        "id": 12,
```

```
            "url": "http://localhost:5000/api/v1/fileGroup/12/",
            "document": {
                "id": 12,
                "url": "http://localhost:5000/api/v1/document/12/",
                "name": "Test Baustellen.kml",
                "description": "Testdaten (Originalformat: KML)",
                "file_groups": "http://localhost:5000/api/v1/document/12/filegroup/",
                "private": false,
                "owner": {
                    "id": 1,
                    "username": "testuser",
                    "first_name": "",
                    "last_name": ""
                },
                "created_at": "2015-05-29T14:21:55.026231Z",
                "preview": "http://localhost:5000/api/v1/document/12/preview/",
                "type": "document"
            },
            "files": [
                {
                    "id": 18,
                    "url": "http://localhost:5000/api/v1/file/18/",
                    "file_name": "Baustellen.kml",
                    "file_format": "KML",
                    "file_group": "http://localhost:5000/api/v1/fileGroup/12/"
                }
            ],
            "urls": [],
            "data": "http://localhost:5000/api/v1/fileGroup/12/data/",
            "preview": "http://localhost:5000/api/v1/fileGroup/12/preview/",
            "related_transformations": [
                {
                    "id": 2005,
                    "name": "Sanitize Baustellen.kml"
                }
            ]
        },
        ...
]
```

Siehe auch **??**.

File Group-Details - GET /api/v1/fileGroup/<id>/

Liefert Detail-Informationen zu einer File Group.

Programmcode 5.12: Beispielantwort für GET /api/v1/fileGroup/12/

```
{
      "id": 12,
      "url": "http://localhost:5000/api/v1/fileGroup/12/",
```

```
        "document": {
            "id": 12,
            "url": "http://localhost:5000/api/v1/document/12/",
            "name": "Test Baustellen.kml",
            "description": "Testdaten (Originalformat: KML)",
            "file_groups": "http://localhost:5000/api/v1/document/12/filegroup/",
            "private": false,
            "owner": {
                "id": 1,
                "username": "testuser",
                "first_name": "",
                "last_name": ""
            },
            "created_at": "2015-05-29T14:21:55.026231Z",
            "preview": "http://localhost:5000/api/v1/document/12/preview/",
            "type": "document"
        },
        "files": [
            {
                "id": 18,
                "url": "http://localhost:5000/api/v1/file/18/",
                "file_name": "Baustellen.kml",
                "file_format": "KML",
                "file_group": "http://localhost:5000/api/v1/fileGroup/12/"
            }
        ],
        "urls": [],
        "data": "http://localhost:5000/api/v1/fileGroup/12/data/",
        "preview": "http://localhost:5000/api/v1/fileGroup/12/preview/",
        "related_transformations": [
            {
                "id": 2005,
                "name": "Sanitize Baustellen.kml"
            }
        ]
    }
```

Document Das Dokument, zu dem diese File Group gehört. Siehe auch **????**.

Files Führt die zur File Group gehörenden Dateien auf. Siehe auch **??**.

URLs Führt die zur File Group gehörenden URLs auf. Siehe auch **??**.

Data URL für den Download der Daten. Siehe auch **??**.

Preview URL für eine Vorschau der Daten in dieser File Group. Siehe auch **??**.

Related Transformations Führt Transformationen auf, die diese File Group verwenden.

5.5.9 Files

File-Liste - GET /api/v1/files/

Liefert eine Liste der vorhandenen Dateien.

Programmcode 5.13: Beispielantwort für GET /api/v1/file/

```
{
    "count": 41,
    "next": "http://localhost:5000/api/v1/file/?page=2",
    "previous": null,
    "results": [
        {
          "id": 1,
          "url": "https://opendatahub-hsr-dev.herokuapp.com/api/v1/file/1/",
          "file_name": "mockaroo.com.csv",
          "file_format": "CSV",
          "file_group":
           ↪  "https://opendatahub-hsr-dev.herokuapp.com/api/v1/fileGroup/1/"
        },
        ...
    ]
}
```

Dieser Endpunkt unterstützt Paging (**??**).

File-Details - GET /api/v1/file/<id>

Liefert Informationen zu einer einzelnen Datei.

Programmcode 5.14: Beispielantwort für GET /api/v1/file/1/

```
{
    "id": 1,
    "url": "https://opendatahub-hsr-dev.herokuapp.com/api/v1/file/1/",
    "file_name": "mockaroo.com.csv",
    "file_format": "CSV",
    "file_group": "https://opendatahub-hsr-dev.herokuapp.com/api/v1/fileGroup/1/"
}
```

5.5.10 URLs

URL-Liste - GET /api/v1/url/

Liefert eine Liste der vorhandenen URLs.

Programmcode 5.15: Beispielantwort für GET /api/v1/url/

```
{
    "count": 2,
```

```
    "next": null,
    "previous": null,
    "results": [
        {
            "id": 1,
            "url": "http://localhost:5000/api/v1/url/1/",
            "source_url": "http://maps.zh.ch/wfs/HaltestellenZHWFS",
            "url_format": null,
            "refresh_after": 3600,
            "type": "wfs",
            "file_group": "http://localhost:5000/api/v1/fileGroup/1001/"
        },
        ...
    ]
}
```


Dieser Endpunkt unterstützt Paging (**??**).

URL-Liste - GET /api/v1/url/1/

Liefert Detail-Informationen zur URL.

Programmcode 5.16: Beispielantwort für GET /api/v1/url/1/


```
{
    "id": 1,
    "url": "http://localhost:5000/api/v1/url/1/",
    "source_url": "http://maps.zh.ch/wfs/HaltestellenZHWFS",
    "url_format": null,
    "refresh_after": 3600,
    "type": "wfs",
    "file_group": "http://localhost:5000/api/v1/fileGroup/1001/"
}
```


5.5.11 Hilfs-Views

Diese Views sind kein direkter Bestandteil des REST APIs, dienen jedoch als Unterstützung dazu.

ODHQL Checker - POST /api/v1/parse

Parsed eine ODHQL Abfrage und liefert die Fehlermeldungen in einem zur Fehlersuche hilfreichen Format.

Benutzerdokumentation - GET /api/v1/odhql/doc/

Erstellt die Benutzerdokumentation anhand der Inline-Dokumentation im Code und liefert das Resultat als HTML-Seite zurück.

5.6 User Interface

5.6.1 Authentifizierung

Zur Authentifizierung der Benutzer wird der OAuth 2 Standard verwendet. Dieser erlaubt ein Login mit einem GitHub, Facebook oder anderem Konto. Implementiert wurde dieser Standard auf der REST Seite mit Hilfe von verschiedenen Django Plugins. Beispielsweise wurde python-social-auth verwendet um den Benutzer gegenüber diesen Diensten zu authentisieren. Die REST Schnittstelle wird dann über einen Token gesichert.

5.6.2 Transformationen erstellen

Wir wollten das Erstellen einer Transformation so einfach wie möglich gestalten. Zu diesem Zweck haben wir uns früh dazu entschlossen einen Wizard zu implementieren, welcher die Erstellung einer Transformation vereinfachen soll (**??**). In den frühen Phasen des Projektes war es notwendig den Wizard zu verwenden. In der gelieferten Beta-Version, ist der Wizard optional. Die Transformation kann auch über den manuellen Editor erstellt werden (**??**), dann müssen allerdings entweder die Tabellennamen bekannt sein, oder es kann nur ein Template erstellt werden.

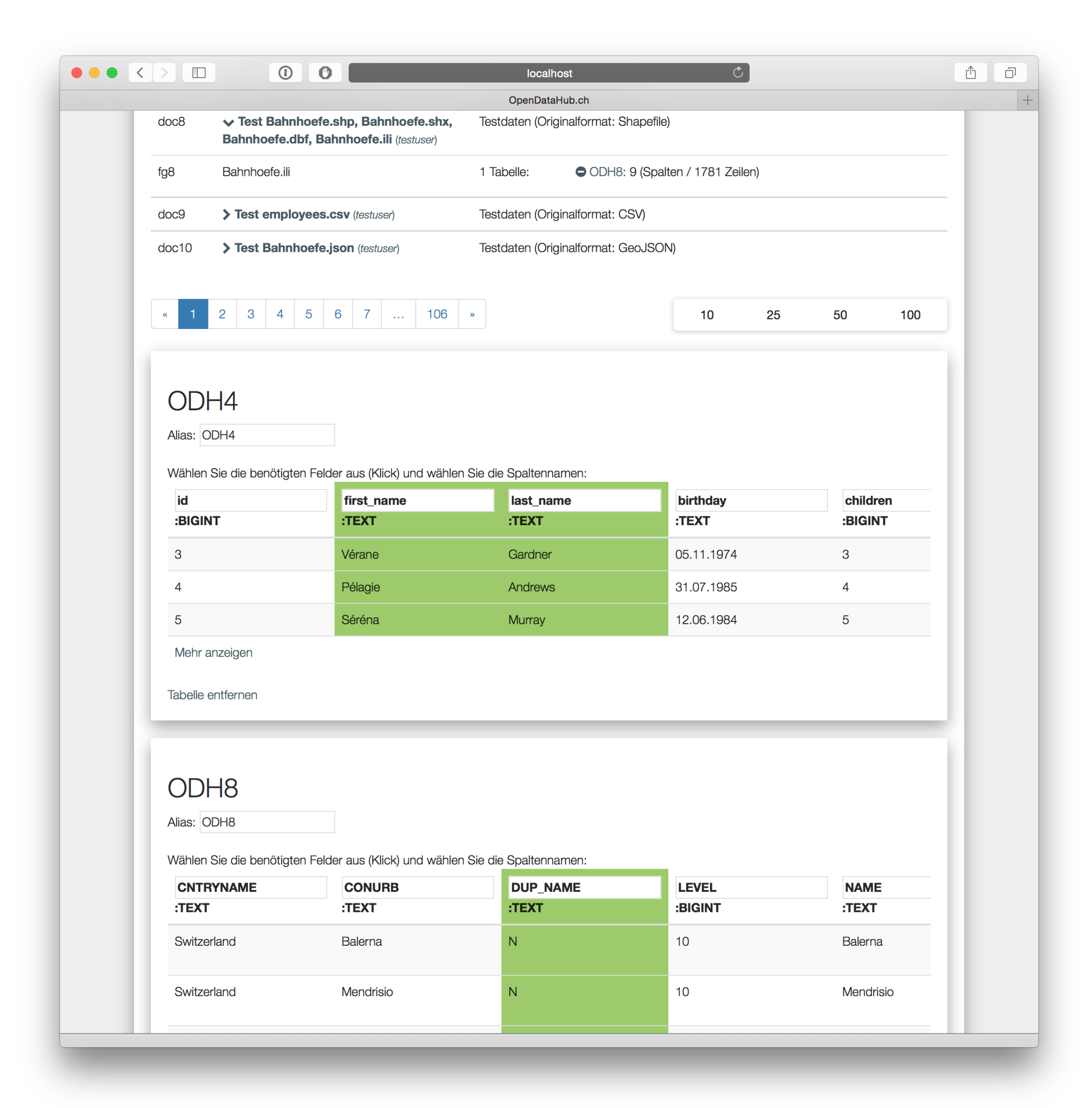


Abbildung 5.11: Tabellen können per Klick verbunden werden

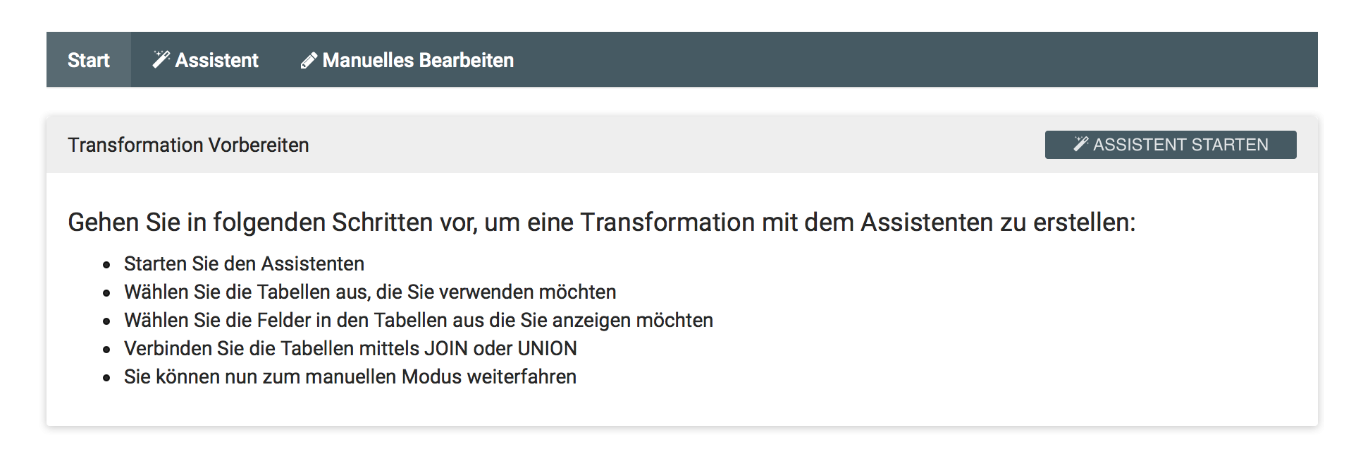


Abbildung 5.12: Einleitung in den Wizard

In einem ersten Schritt kann eine Transformation grundlegend erstellt werden. Einfache JOIN und UNION Operationen werden vom Benutzer konfiguriert.

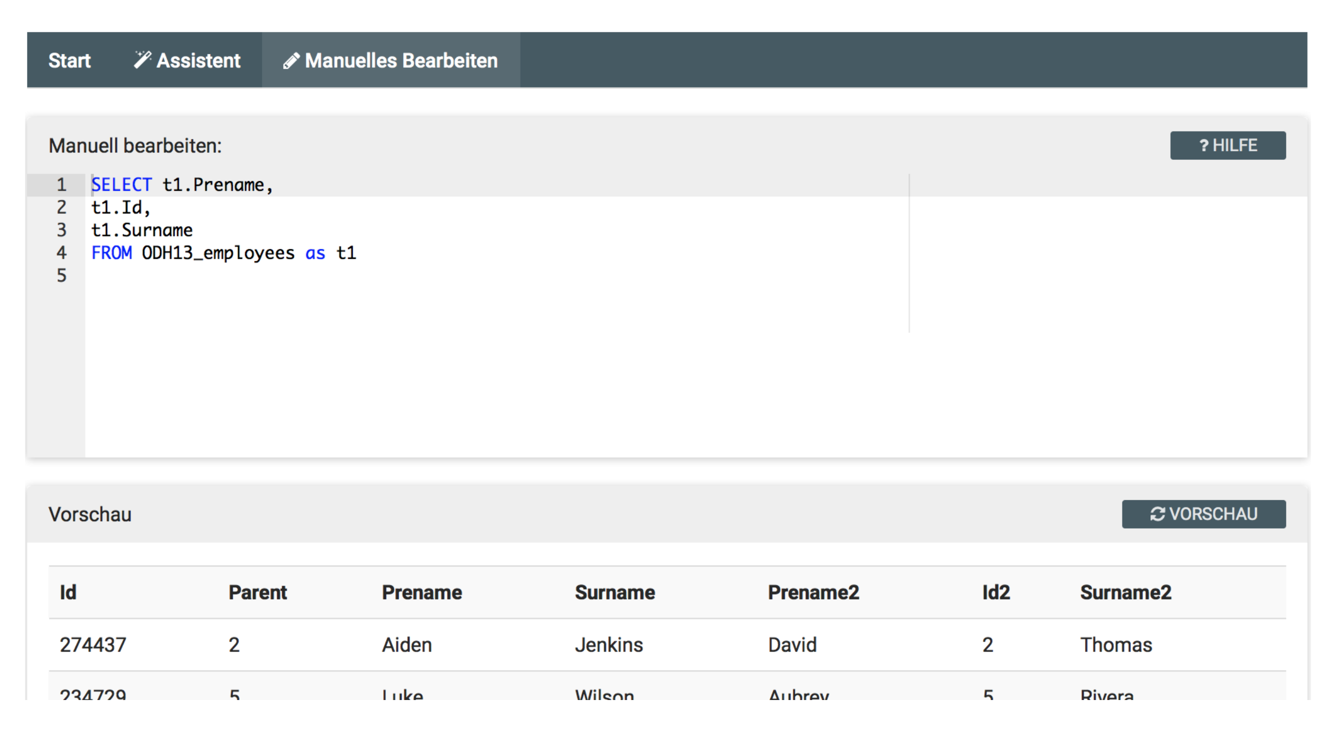


Abbildung 5.13: Transformation manuell bearbeiten

Später können dann die komplexeren Operationen auf die Daten angewendet werden. Hierfür wird der manuelle Modus, in **??** beschrieben, verwendet. Im manuellen Modus steht eine Hilfe Seite zur Verfügung, welche sämtliche erweiterten Operationen erklärt.
Wird eine Transformation "als Template" verwendet, wird beim Speichern keine Prüfung der zugrunde liegenden Daten durchgeführt. So können syntaktisch korrekte, aber noch mit fehlenden Daten belastete Transformationen erstellt werden.

5.6.3 Transformationen bearbeiten

Transformationen können entweder geklont oder überschrieben werden. **??** zeigt eine bestehende Transformation, die der angemeldete Benutzer erstellt hat. Beim Klonen legt der Benutzer eine neue Transformation aufgrund der Daten einer bestehenden Transformation an. Beim Überschreiben kann der Besitzer einer Transformation diese nochmals verändern. Es kann bei diesen Operationen allerdings kein Wizard mehr verwendet werden, da dieser alle manuellen Änderungen überschreiben würde.

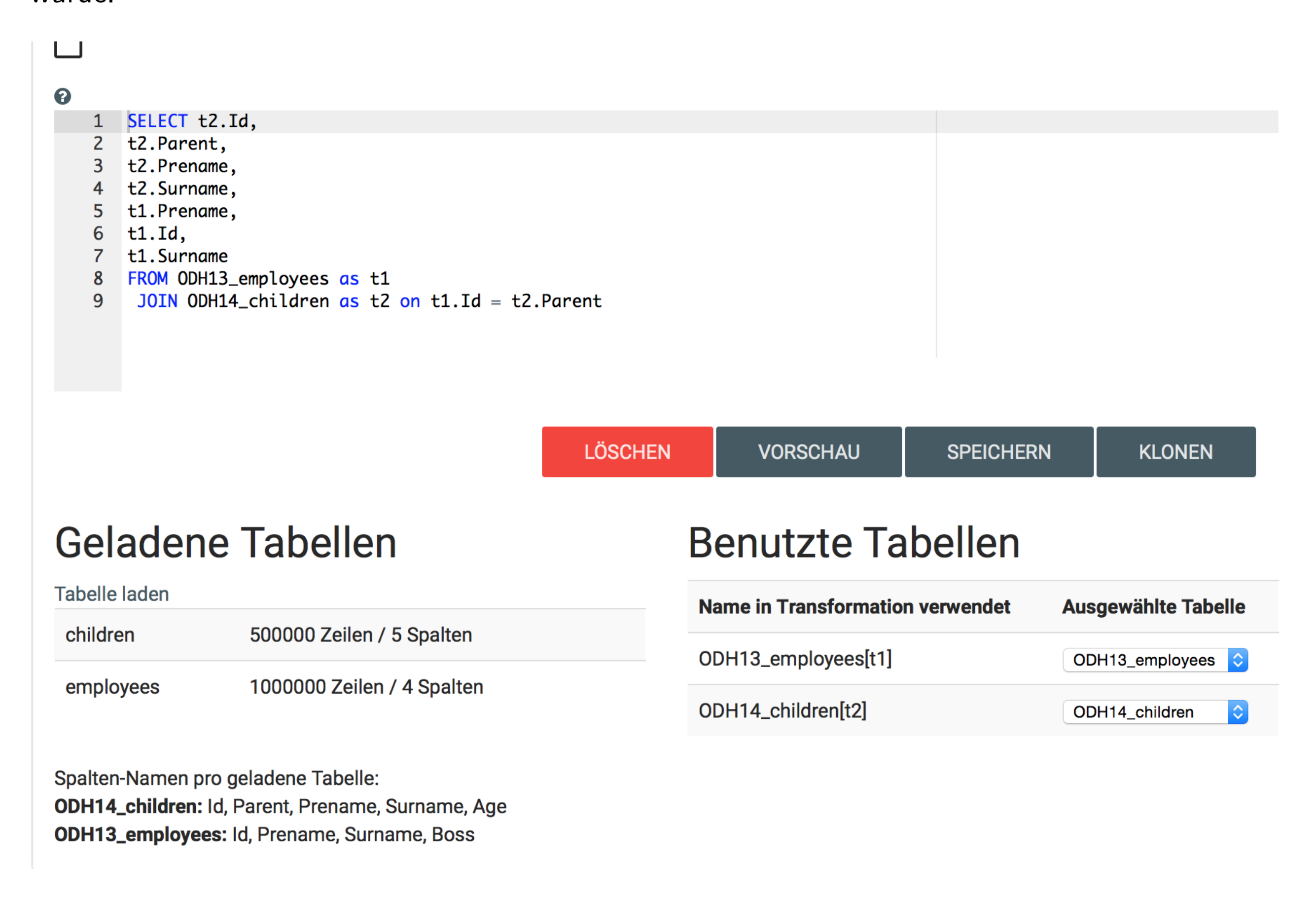


Abbildung 5.14: Bestehende Transformation editieren

6 Testing

6.1 User Interface

Bei allen folgenden Tests wird angenommen, dass der Benutzer die Web-Applikation bereits geladen hat. Für die Tests wird entweder https://dev.opendatahub.ch oder eine lokale Instanz (erreichbar unter http://localhost:5000/) verwendet.
Als Testpersonen stellten sich zur Verfügung:

1. Philipp Christen

6.1.1 UI01: Anmelden

Anmeldestatus Nicht angemeldet.

Aufgabe Der Benutzer will sich anmelden.

Walkthrough

1. Der Benutzer klickt auf “Anmelden/Registrieren”.
2. Der Benutzer meldet sich mit einem der angebotenen OAuth-Providern an.
3. Ein Pop-Up mit dem normalen Login UI des Providers erscheint. Der Benutzer meldet sich an.
   3.a. Die Anmeldung beim Provider schlägt fehl. Der Anmelde-Vorgang wird abgebrochen.
4. Nach kurzer Wartezeit ist der Benutzer bei OpenDataHub angemeldet.

Testperson 1

1. Der Benutzer sieht den “Anmelden”-Button und klickt darauf.
2. Da er einen GitHub-Account hat und dort bereits angemeldet ist, wählt er diesen Provider.
3. Die Anmeldung erfolgt problemlos. Der Benutzer wünscht sich anschliessend zu sehen, wer eingeloggt ist (Name fehlt).

6.1.2 UI02: Daten bereitstellen

Anmeldestatus Angemeldet.

Aufgabe Der Benutzer will die Daten aus der Datei "Baustellen Mai 2015.xls" mit einer Liste von Baustellen im Appenzell Ausserrhoden anbieten.

Walkthrough

1. Der Benutzer klickt auf "Teilen".
2. Der Benutzer füllt die Felder "Name/Titel" sowie "Beschreibung" aus.
   Zum Beispiel: Name: "Baustellen Mai 2015", Beschreibung: "Baustellen-Liste Appenzell Ausserrhoden Mai 2015".
3. Die Felder "Privat" sowie "Format" werden ignoriert.
   - 3.a. Die Daten sollen privat bleiben. Der Benutzer klickt das Feld "Privat" an.
   - 3.b. Der Benutzer will sicherstellen, dass das Format richtig erkannt wird und wählt aus der Liste das Format "Excel" aus.
4. Da die Daten als Datei vorliegen wählt der Benutzer "Dateien hochladen" aus.
5. Anschliessend zieht er die Datei von einem File-Browser direkt auf das Upload-Feld.
6. Ein Klick auf den Button "Teilen" lädt die Datei hoch. Eine grün hinterlegte Nachricht "Ihre Daten wurden gespeichert" erscheint.
   - 6.a. Die Daten liegen in einem Format vor, welches von OpenDataHub nicht erkannt wird. Es wird eine rot hinterlegte Fehlermeldung angezeigt.
7. Nach kurzer Zeit wird der Benutzer auf die Detail-Ansicht seiner Daten weitergeleitet und sieht eine Vorschau seiner Daten.

Testperson 1

1. Der Benutzer interpretiert "Anbieten" korrekt als "Teilen" und klickt den entsprechenden Menu-Eintrag an.
2. Als Format belässt er "Auto", da er "faul" sei.
3. Bei der Datei-Auswahl bemängelt der Benutzer, dass der Datei-Name abgeschnitten wird, obwohl noch genug Platz vorhanden ist.
4. Beim Speichern freut er sich über die Toast-Message ("grün!"), liest den Text jedoch nicht durch.
5. Nach erfolgreichem Speichern ist der Benutzer unsicher, ob die Aufgabe abgeschlossen ist und sucht nach Share-/Social-Buttons.

6.1.3 UI03: Daten transformieren (Mit Assistent)

Anmeldestatus Angemeldet.

Aufgabe Der Benutzer interessiert sich für Baustellen auf Zürcher Kantonsstrassen. Er will Gemeinde, Baustellen-Status, Beginn und Ende der Bauarbeiten sowie den Strassennamen, jeweils mit menschenlesbaren Feld-Bezeichnungen, und erstellt dazu eine Transformation.
Der Benutzer hat keine Erfahrungen mit ODHQL oder SQL und verwendet den Assistenten.

Walkthrough

1. Der Benutzer klickt auf "Neue Transformation".
2. In typischer Benutzer-Manier wird die Anleitung auf der Einstiegs-Seite ignoriert.
3. Der Benutzer klickt auf "Assistent".
4. Die gewünschten Daten sind nicht auf der ersten Seite zu finden. Daher verwendet der Benutzer das Suchfeld.
   4.a. Der Benutzer übersieht das Suchfeld und klickt durch die Seiten bis er die gewünschten Daten findet.
5. Nach erfolgreicher Suche klickt der Benutzer auf den Namen des Packages (Dokument/Transformation).
6. Ein Klick auf den Tabellen-Namen "ODH18_baustellen_detailansicht" fügt die Tabelle zur Abfrage hinzu.
7. Der Benutzer wählt die in der Aufgabe erwähnten Felder durch einen Klick auf "Feld hinzufügen" aus.
8. Durch bearbeiten der Text-Felder in den Spalten-Titeln benennt der Benutzer die Felder um.
9. Anschliessend vergibt er einen Namen sowie eine Beschreibung und klickt auf den "Speichern"-Button.
10. Eine grüne Erfolgsmeldung erscheint und nach kurzer Zeit wird der Benutzer zur Detail-Ansicht der neuen Transformation weitergeleitet.

Testperson 1

1. Nach kurzer Orientierung in der Benutzeroberfläche findet der Benutzer das Suchfeld.
2. Die Suche nach "baustelle" ist erfolgreich. Der Benutzer ist unsicher, ob "baustellen_detailansicht" oder "baustellen_uebersicht" richtig ist und wird angewiesen, die "Detail"-Version auszuwählen. Dies ist eine Unschönheit im WFS von dem die Daten stammen und kein UI-Problem.
3. Er ist unsicher, wo er klicken muss um die Felder hinzuzufügen. Nach kurzer Orientierung findet er die richtige Stelle (alles ausser Alias-Feld).
4. Bei der Eingabe der Aliases ist der Benutzer enttäuscht, dass die Tabelle nicht automatisch weiter scrollt.
5. Das Speichern der Transformation funktioniert problemlos.

6.1.4 UI04: Daten transformieren (Ohne Assistent)

Anmeldestatus Angemeldet.

Aufgabe Der Benutzer interessiert sich für Baustellen auf Zürcher Kantonsstrassen. Er will Gemeinde, Baustellen-Status, Beginn und Ende der Bauarbeiten, den Strassennamen sowie die Geometrie als Punkt, jeweils mit menschenlesbaren Namen, und erstellt dazu eine Transformation.
Das Datums-Format soll dem Format "Tag. Monat. Jahr" (z.B. "01.01.1970") entsprechen.
Da er bereits Erfahrung mit ODHQL oder SQL hat schreibt er die Transformation selbst. Der Assistent kann als Startpunkt verwendet werden, unterstützt jedoch nicht alle benötigte Funktionalität.
Hinweis: Die Sprache ODHQL lehnt sich stark an SQL an. Ausserdem ist in der Hilfe eine ODHQL-Referenz enthalten.

Walkthrough

1. Der Benutzer klickt auf "Neue Transformation" und liest sich den Informations-Text durch.
2. Um die Liste der Spalten zu erhalten verwendet er den Assistenten (siehe **??**, Schritte **??**–**??**).
3. Durch Klick auf "Manuelles Bearbeiten" wechselt der Benutzer in den Editier-Modus.
4. Mit Hilfe der ODHQL-Referenz ergänzt der Benutzer die fehlenden Funktionen (ST_Centroid, To_Date, To_Char). Zur Überprüfung der Abfrage verwendet der Benutzer periodisch den "Vorschau"-Knopf.
5. Anschliessend vergibt er einen Namen sowie eine Beschreibung und klickt auf den "Speichern"-Button.
6. Eine grüne Erfolgsmeldung erscheint und nach kurzer Zeit wird der Benutzer zur Detail-Ansicht der neuen Transformation weitergeleitet.
   6.a. Die Transformation ist fehlerhaft. Eine rote Fehlermeldung zeigt detailliertere Informationen zum Problem an.

Testperson 1

1. Der Benutzer verwendet den Assistenten als Startpunkt und will anschliessend in den Editier-Modus wechseln. Dazu muss er erst nach oben scrollen, was er als "mühsam" bezeichnet.
2. Das Hilfe-Icon funktioniert nicht. Die Punkt "Hilfe" im Menu schon, öffnet die Hilfe aber im selben Browser-Fenster. Der Benutzer muss die Abfrage nochmals neu zusammen klicken.
3. Nach der Eingabe eines fehlerhaften Queries stellt der Benutzer fest, dass die Fehlermeldung ungünstig platziert ist (unterhalb der Vorschau).
4. Der Benutzer hat einige Schwierigkeiten mit der Erstellung des Queries. Er hat zwar die Vorlesung Dbs1 an der HSR besucht, verwendet SQL aber sehr selten. Mit Hilfe der Dokumentation gelingt es schliesslich doch.

6.1.5 UI05: Daten beziehen

Anmeldestatus Nicht angemeldet.

Aufgabe Um die Daten in seinem System weiterverwenden zu können will der Benutzer die Resultate der vorhin erstellten Transformationen als GML-Datei herunterladen.

Walkthrough

1. Der Benutzer sucht die Transformation in der Daten-Liste und klickt auf den Namen.
2. Ein Klick auf "Herunterladen" öffnet die Format-Auswahl.
3. Die Auswahl von "GML" führt zu einem erfolgreichen Download der Daten.

Testperson 1 Die Erfüllung der Aufgabe stellte kein Problem dar.
Nach Abschluss des Tests experimentierte der Benutzer ein wenig mit exotischeren Unicode-Namen. Namen komplett ohne Zeichen im ASCII-Bereich stellten ein Problem dar - der Download schlägt in diesen Fällen fehl.

6.2 Unit Tests

Alles in allem wurden über 300 Unit-Tests erstellt. Diese decken die im folgenden beschriebenen Bereiche ab.

6.2.1 Formate

Testet ob eine Serie von Dateien richtig identifiziert werden.

6.2.2 Parser

Überprüft ob eine Serie von Dateien richtig gelesen werden kann, und ob wie erwartet eine Geometrie-Spalte vorhanden ist oder nicht.

6.2.3 Formatter

Formatiert eine Serie von DataFrames und überprüft, ob die erwartete Anzahl Dateien erzeugt wurde.

6.2.4 Fixtures

Mit "loadfixtures" kann eine Serie von Test-Daten, auch Fixtures genannt, geladen werden. Diese Tests überprüfen, ob die geladenen Fixtures korrekt geparsed und anschliessend in jedem bekannten Format formatiert werden können. Dabei wird dasselbe API verwendet, wie auch von Benutzern zum herunterladen von Dateien verwendet wird.
Diese Tests werden dynamisch generiert.

6.2.5 Ogr2ogr

Versucht, mittels der ogr2ogr-Hilfsklasse eine Serie von Daten zu konvertieren.

6.2.6 ODHQL: Parser

Überprüft, ob diverse Syntax-Elemente korrekt geparsed werden.

6.2.7 ODHQL: Functions

Testet die Implementation diverser ODHQL-Funktionen.

6.2.8 ODHQL: Interpreter

Tests für den Interpreter.

6.2.9 REST API

Testet einige API-Funktionen.

7 Resultate und Ausblick

7.1 Resultate

Die Resultate der Arbeit sind in **??** beschrieben.

7.2 Weiterentwicklung

In diesem Abschnitt werden diverse Möglichkeiten zur Weiterentwicklung von OpenDataHub aufgezeigt.

7.2.1 GDAL 2.0

Diverse Treiber[1] wurden in GDAL 2.0 massiv verbessert bzw. richtig implementiert. Sobald ein Release vorhanden ist, sollte ein Upgrade in Betracht gezogen werden.

7.2.2 Native Unterstützung für via ogr2ogr implementierte Formate

Die Unterstützung für die Formate Interlis 1, GML und WFS (nur Parser) ist via ogr2ogr implementiert. Da ogr2ogr auf Datei-Ebene operiert, müssen die Daten über ein Zwischenformat konvertiert werden. Für den Formatter sieht das wie folgt aus:

$$\text{DataFrame in ODH} \rightarrow \text{Zwischenformat} \rightarrow \text{ogr2ogr} \rightarrow \text{Zielformat}$$

Analog für den Parser. Beide Konvertierungs-Schritte sind potentiell fehlerbehaftet, ausserdem sind nicht alle ogr2ogr-Treiber von gleicher Qualität.
Daher wäre es wünschenswert, diese Formate mit nativen Parsern und Formattern zu unterstützen.

7.2.3 Ausbau der Webapplikation

Nach Untersuchung der bereits vorhandenen Produkte wurde entschieden, die Webapplikation minimal zu halten. Die Konvertierung und Transformation von Daten hatte Priorität.
Wir sehen folgende Optionen:

- Ausbau der verwalteten Metadaten
- Mutation der Dateien, die zu einem Package gehören
- Diskussion von Daten/Transformationen, evtl. Issue-Tracker
- Integration von CKAN

[1] Interlis 1/2, GeoPackage

7.2.4 Unterstützung von verschiedenen Encoding-Optionen

Sofern vom Datei-Format nichts anderes vorgegeben wird, lesen die implementierten Parser allen Input mit UTF-8 als Encoding. Gerade bei Formaten wie CSV, welche von Benutzern direkt erstellt werden können, z.B. mit Excel, ist dies nicht immer optimal.
Wir sehen folgende Optionen:

- Dem Benutzer eine Angabe des verwendeten Encodings ermöglichen
- Eine Encoding-Detection durchführen, z.B. mit chardet[2]. Dies funktioniert in vielen Fällen einigermassen gut, ist jedoch nicht zuverlässig.

Am besten wäre wohl eine Kombination der beiden Optionen.

7.2.5 Zusätzliche ODHQL-Funktionen

Bisher sind zwar diverse Funktionen implementiert, gerade im Geo-Bereich gibt es jedoch noch einiges an Erweiterungspotential.

7.2.6 Datentypen

ODHQL unterstützt bisher folgende Datentypen: Integer (32 Bit), SmallInt (16 Bit), BigInt (64 Bit), Float (64 Bit), DateTime (64 Bit), Boolean, Text und Geometry. (Date-/Time-)Interval wird theoretisch unterstützt, es gibt jedoch aktuell keine Funktionen welche diesen Typ verwenden.
Mögliche Erweiterungen:

- Wertebereiche für bereits bestehende Datentypen, z.B. Längenbeschränkung für Text, oder Boundaries für Geometrien
- Subqueries bzw. Nested Tables
- Baumstrukturen, Key-Value-Pairs

Ausserdem sollte untersucht werden, ob Geodaten in allen Fällen korrekt behandelt werden.

7.2.7 Breitere Formatunterstützung

Nach Bedarf können weitere Formate implementiert werden.

7.2.8 ODH als WFS-Server

ODH unterstützt WFS als Daten-Quelle. Die Idee ist, dass eine API implementiert wird, welche eine Datenquelle oder WFS-URL und optional eine ODHQL-Abfrage entgegen nimmt, die Transformation durchführt und anschliessend das Resultat als WFS anbietet.

[2] https://pypi.python.org/pypi/chardet

7.2.9 Performanz bei grossen Datenmengen

Im Rahmen der Arbeit wurden Dokumente mit Grössen bis ca. 100 MB (ca. 1 Mio. Records) mit guten Resultaten getestet. Im Gespräch mit Herrn Dorfschmid wurde festgestellt, dass dies für viele Anwendungen ausreicht.
Falls massiv grössere Datenmengen verarbeitet werden müssen ist die aktuelle Architektur aber suboptimal und stark limitiert durch den verfügbaren Speicher. In diesem Falle sollte evtl. auf eine streambasierte Version gewechselt werden.
Zu beachten ist, dass diverse verwendete Bibliotheken (unter anderem auch GDAL) nicht streamorientiert arbeiten.

7.2.10 Unterstützung für weitere Datenstrukturen

Die aktuelle Version von ODH unterstützt nur tabellarische Daten. Dies könnte um Unterstützung für baumartige Daten, wie zum Beispiel XML, erweitert werden.

7.2.11 Unterstützung für weitere Transformationsarten

Ogr2ogr ermöglicht es, Transformationen per SQL in einer Datenbank auszuführen. Dies könnte auch in ODH implementiert werden, würde jedoch zu Performance-Einbussen führen.

7.2.12 Bereinigung des REST API

Das REST API funktioniert, beinhaltet jedoch einige Inkonsistenzen bzw. Unschönheiten.
Ein Beispiel: Bei Anfragen auf /api/v1/fileGroup/ wird das zur File Group gehörende Dokument zurückgeliefert. /api/v1/document/<id>/filegroup/ liefert alle zu einem Dokument gehörenden File Groups. Auch hier enthält die Antwort das Dokument, da derselbe Serializer verwendet wird. Dies führt jedoch dazu, dass bei Dokumenten mit mehr als einer File Group die Dokument-Informationen mehrfach in der Antwort enthalten sind.
Stattdessen sollte in dieser Situation nur die Dokument-URL geliefert werden.

7.2.13 Templates in neuen Transformationen verwenden können

Anstatt ein Template zuerst als Transformation abzuspeichern, könnte es sinnvoll sein, ein Template direkt in einer neuen Transformation verwenden zu können.

7.2.14 In Templates Spaltennamen überschreiben

Im Moment können im OpenDataHub die Tabellennamen generalisiert werden. Wenn ein Template andere Spaltennamen als das referenzierte Dokument verwendet, können diese nicht automatisch ausgetauscht werden.

7.2.15 Kategorien und Tags für Dokumente und Transformationen

In einer weiteren Version könnten Dokumente und Transformationen mit Tags und Kategorien versehen werden. Dies würde die Übersichtlichkeit erhöhen.

7.2.16 Benutzer-Gruppen mit unterschiedlichen Rechten

Zu Beginn der Arbeit wurde zusammen mit dem Betreuer entschieden, dass die Benutzer-Verwaltung minimal gehalten werden soll. Insbesondere sollte kein Aufwand zur Administration der Benutzer anfallen.

In Zukunft könnte es wünschenswert sein, Benutzergruppen mit unterschiedlichen Rechten einzuführen, z.B. eine Unterscheidung zwischen normalen Nutzern und Administratoren.

Teil III

Projektmanagement

1 Vorgehensmodell

Als Vorgehensmodell für die Bachelorarbeit wurde Scrum verwendet. Scrum ist ein agiles, iteratives Modell, welches sich aufgrund dessen Einfachheit und nicht besonders strikter Vorgaben sehr gut für kleinere, gut überschaubare Projekte eignet. Scrum basiert wie viele andere agile Projektmanagement-Methoden auf den vier im Agilen Manifest [**agilemanifesto**] beschriebenen Grundsätzen:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Wobei der Punkt »Working software over comprehensive documentation« aufgrund der Tatsache, dass es sich um eine akademische Arbeit handelt, vernachlässigt wird.

1.1 Rollen

Rolle	Beschreibung
Product Owner	Der Product Owner bestimmt die Anforderungen (User Stories) und priorisiert diese. Die Priorität gibt an, welche User Stories es im nächsten Sprint zu erledigt gilt.
Team	Das Team besteht aus allen für das Projekt tätigen Entwicklern.
Scrum Master	Der Scrum Master kümmert sich um die fortlaufende Kontrolle des Fortschritts sowie Optimierungsmöglichkeiten. In der Praxis ist er zudem für die Leitung der Daily Scrum Meetings zuständig.

Tabelle 1.2: Scrum Projektrollen

1.2 Artefakte

Artefakt	Beschreibung
Product Backlog	Der Product Backlog enthält alle noch nicht abgeschlossenen User Stories des Projekts. Vor Beginn des neuen Sprints werden die noch offenen User Stories vom Product Owner repriorisiert und dessen Aufwand vom Team geschätzt.
Sprint Backlog	Der Sprint Backlog enthält alle für einen bestimmten Sprint ausgewählten User Stories. Das Team setzt die Quantität fest und der Product Owner bestimmt den Inhalt und somit welche User Stories in den nächsten Sprint einfliessen.
Burndown Charts	Mittels Jira werden laufend die Ist- und Sollstunden des Sprints addiert und daraus ein regressives Diagramm erstellt. Dies soll aufzeigen wie man zeitlich steht, um mögliche Zeitprobleme zu vermeiden.

Tabelle 1.4: Scrum Artefakte

1.3 Sprints

Die Iterationen, bei Scrum "Sprint" genannt und in **??** dargestellt, dauerten mit Ausnahme des ersten und letzten Sprints jeweils zwei Wochen. So konnte, wie bei Scrum üblich, zum Ende jedes Sprints ein Deliverable, sei es eine lauffähige Applikation oder ein anderes Resultat, zur Verfügung gestellt werden.

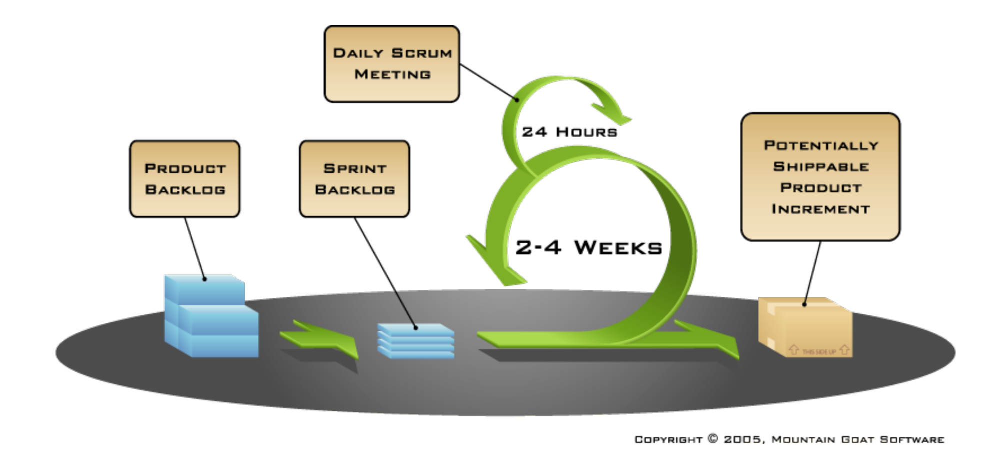


Abbildung 1.1: Scrum Sprint [**scrum**]

2 Rollen und Verantwortlichkeiten

2.1 Prof. Stefan Keller



Prof. S. Keller, Mitarbeiter des Institut für Software (IFS) und Dozent an der HSR übernimmt im Projekt eine Doppelrolle:

- Als Experte bzw. Betreuer übernimmt er die Aufsicht und Bewertung der Bachelorarbeit
- und als fiktiver Kunde die Kontaktperson zur Aufnahme der Anforderungen (Product Owner).

2.2 Remo Liebi



Remo Liebi, Teilzeitstudent an der HSR, Inhaber der IT-Firma liebi.net, ist Teil des Entwicklungsteams.

2.3 Christoph Hüsler



Christoph Hüsler, Vollzeitstudent an der HSR, JavaEE-Entwickler und Datenbank-Experte bei Xinventa GmbH, ist Teil des Entwicklungsteams.

2.4 Fabio Scala



Fabio Scala, Teilzeitstudent an der HSR und Banking Software Engineer bei Swisscom (Schweiz) AG, ist Teil des Entwicklungsteams und übernimmt zusätzlich die Rolle des Scrum Masters.

3 Risiken

Für die Abschätzung der Projektrisiken wurde auf existierende [**risk**] Risikochecklisten gesetzt, um möglichst keine potenziellen Risiken bei der Analyse auszulassen. Die sehr umfangreiche Risikoliste von **Wallace:2004:SPR:975817.975819** [**Wallace:2004:SPR:975817.975819**] wurde mit den bekannten Top Zehn Software-Projektrisiken aus dem Jahre **boehm** [**boehm**] sowie mit eigenen, projektspezifischen Risiken ergänzt. Auch aus der Sicht des Teams für das Projekt nicht relevante Risiken wurden protokolliert und gelten als Nachweis, dass auch diese miteinbezogen und nicht etwa ignoriert oder gar vergessen wurden.

ID	Kategorie	Risiko	Schadens-potenzial	Eintritts-wahrschei-nlichkeit	Schaden	Präventionsmassnahme	Massnahme bei Eintritt
R01	**Mensch**	Teammitglied sträubt sich gegen Änderungen	2	1	2	Frühzeitig beim Betreuer (Prof. S. Keller) melden	-
R02		Kon ikte im Team	3	2	6	Frühzeitig beim Betreuer (Prof. S. Keller) melden	-
R03		Negative Einstellung im Team	3	1	3	Frühzeitig beim Betreuer (Prof. S. Keller) melden	-
R04		Fehlende Motivation im Team	3	1	3	Frühzeitig beim Betreuer (Prof. S. Keller) melden	-
R05		Fehlende Kooperation im Team	2	1	2	Frühzeitig beim Betreuer (Prof. S. Keller) melden	-
R06	**Anforderungen**	Ständig ändernde Anforderungen	1	2	2	Agile Vorgehensweise. Häu ge Projektsitzungen mit dem Kunden	Agile Vorgehensweise erlaubt Anpassungen und Repriorisierung der Anforderungen
R07		Anforderungen ungenügend erfasst	1	2	2	Agile Vorgehensweise. Häu ge Projektsitzungen mit dem Kunden	Agile Vorgehensweise erlaubt Anpassungen und Repriorisierung der Anforderungen
R08		Unklare Anforderungen	2	3	6	Agile Vorgehensweise. Häu ge Projektsitzungen mit dem Kunden	Agile Vorgehensweise erlaubt Anpassungen und Repriorisierung der Anforderungen
R09		Falsche Anforderungen (Umsetzung der falschen Funktionalität)	3	1	3	Agile Vorgehensweise. Am Ende jedes Sprints wird das Resultat mit dem Kunden besprochen.	Change Request als neue User Story in den Product Backlog aufnehmen
R10	**Komplexität**	Verwendung neuer Technologien	4	5	20	In einem ersten Schritt die Technologie (dat) erforschen und die Machbarkeit abschätzen.	Alternative Technologie suchen
R11		Hohe technische Komplexität	3	2	6	Machbarkeit frühzeitig abschätzen	Einbezug externer Experten (z.B. aus dem IFS)
R12		Verwendung unreifer Technologien	4	5	20	In einem ersten Schritt die Technologie (dat) erforschen und die Machbarkeit abschätzen.	Alternative Technologie suchen
R13		Verwendung noch nie zuvor eingesetzter Technologien	2	5	10	In einem ersten Schritt die Technologie (dat) erforschen und die Machbarkeit abschätzen.	Einbezug externer Experten (z.B. dat Entwickler Max Ogden oder dessen IRC Community auf Freenode)
R15	**Planung & Kontrolle**	Fehlende Projektmanagement Plattform	4	1	4	Verwendung von Atlassian JIRA, was sich in der Praxis mehrfach bewährt hat	-
R16		Projektfortschritt wird ungenügend überwacht	4	1	4	Atlassian JIRA bietet umfangreiche Auswertungsmöglichkeiten. Tägliche und Wöchentliche Auswertungen planen.	-
R17		Benötigte Ressourcen oder Aufwand werden falsch abgeschätzt	3	2	6	Burndown Charts zur frühzeitigen Erkennung	Verschieben von User Stories in den Product Backlog. Repriorisierung mit dem Kunden.
R18		Ungenügende Projektplanung	2	2	4	-	-
R19		Meilensteine/Deliverables ungenügend de niert	2	3	6	Umfang der Sprints zu Beginn provisorisch festlegen	Nach Bedarf User Stories repriorisieren und in andere Sprints verschieben
R20		~~Unerfahrene Projektmanager~~	-	0	0	-	-
R21		Ungenügende Kommunikation	3	1	3	Wöchentliche Projektsitzungen. Arbeit vor Or an der HSR. Slack als Kommunikationsplattform.	Zusätzliche xe Meetings innerhalb des Teams.
R22		Gold-plating	3	2	6	Der Kunde setzt die Prioritäten vor jedem Sprint neu.	-
R23		Personalausfall (Krankheit, Unfall, ...)	4	2	8	Höhere Gewalt	Projektscope reduzieren
R24	**Team**	~~Unerfahrene Teammitglieder~~	0	0	0	-	-
R25		Fehlendes Know-how im Team	4	3	12	Frühzeitige Einarbeitung	-
R26	**Organisation (HSR)**	Organisatorische Anpassungen (z.B. Betreuer oder Ansprechperson des Kunden ändert)	5	1	5	Höhere Gewalt, keinen Ein uss.	-
R27		~~Negativer Ein uss durch Corporate Politik~~	-	0	0	-	-
R28		~~Instabile Unternehmensorganisation~~	-	0	0	-	-
R29		~~Firmenorganisation wird umstrukturiert~~	-	0	0	-	-
R30	**Projektspez sich**	Ausfall Infrastruktur eines Teammitglieds (Notebook)	3	0	0	Möglichst Oft die commits auf den Server pushen. Automatisches Backup des Notebooks.	Notebook neu aufsetzen.
R31		dat ist nicht für die Anforderungen des Projekts geeignet	4	2	8	-	Alternative Technologie suchen.
R32		dat ist zu unreif und es fehlt die nötige Dokumentation	3	2	6	Evaluation zu Beginn des Projekts	Einbezug externer Experten (z.B. aus dem IFS)
R33		Das dat Projekt wird nicht weiterentwickelt	3	2	6	Abschätzung durch Grösse der Community, Anzahl Commits, Aktivität	Alternative Technologie suchen oder dat forken und selbst weiterentwickeln.
R34		Der virtuelle Server der HSR fällt temporär aus	2	1	2	Regelmässige Backups	-
R35		Ausfall weiterer Infrastruktur (Git, Slack, ...)	3	1	3	-	Kommunikation via E-Mail.
R36		Es treten technische Hürden auf, die nicht vorhergesehen wurden und nicht zu bewältigen sind	4	1	4	Machbarkeit frühzeitig abschätzen.	Einbezug externer Experten (z.B. aus dem IFS)

Tabelle 3.1: Alle berücksichtigten Risiken

95

3.1 Kritische Risiken

Nach bzw. während der Einarbeitung in dat wurde eine erste Abschätzung der Risiken vorgenommen. Alle irrelevanten oder ausschliessbaren Risiken wurden dabei mit einer Eintrittswahrscheinlichkeit und einem Schadenspotenzial von 0 bewertet.
Nachfolgend werden die als besonders kritisch eingestuften Risiken sowie deren Massnahmen genauer beschrieben. Schon jetzt ist klar, dass das grösste Risiko das Projekt dat selbst darstellt.

Risiko	R02
Titel	Konflikte im Team
Beschreibung	Das Projektteam kennt sich erst seit kurzem und hat davor noch nie zusammengearbeitet. Somit besteht ein erhöhtes Risiko von Meinungsverschiedenheiten während des Projektverlaufs.
Prävention/Massnahme	Frühzeitige Kontaktaufnahme mit dem Betreuer falls die Konflikte nicht intern gelöst werden können.

Risiko	R10, R12, R13
Titel	Verwendung neuer oder unreifer Technologien
Beschreibung	Es werden unreife “bleeding Edge” Technologien verwendet, welche die Entwicklung und Handhabung erheblich erschweren. Siehe auch dat-spezifische Risiken R31, R32 und R33.
Prävention/Massnahme	Es werden vor allem Technologien verwendet die dem Team bereits bekannt sind oder das Risiko durch eine Kurzevaluation reduziert.

Risiko	R31
Titel	dat erfüllt nicht die Anforderungen
Beschreibung	dat und dessen Funktionalität ist nur sehr spärlich beschrieben. Selbst dem Auftraggeber ist der Umfang von dat nicht vollständig bekannt. Es besteht das Risiko, dass dat sich nicht für die Use Cases der Aufgabenstellung eignet.
Prävention/Massnahme	Da es sich bei dieser Arbeit unter anderem spezifisch um eine Evaluation von dat und dessen Möglichkeiten handelt, ist eine Evaluation von alternativen Technologien zunächst zu vermeiden. Durch eine frühzeitige Einarbeitung in dat mit kleineren Prototypen/Use Cases soll die Machbarkeit abgeschätzt werden.

Risiko	R32
Titel	dat ist zu unreif. Fehlende Dokumentation.

Beschreibung	dat besitzt viele unbekannte Fehler und die Dokumentation ist unzureichend. Bereits während des ersten Sprints hat die spärliche Dokumentation von dat zu Bedenken im Team geführt.
Prävention/Massnahme	Dasselbe wie bei R31

Risiko	R33
Titel	dat wird nicht weiterentwickelt
Beschreibung	Der Kernentwickler von dat (Max Ogden) verliert das Interesse am Projekt und dat wird auch nicht von der Community weiterentwickelt.
Prävention/Massnahme	Abschätzung des Risikos durch die Aktivitäten der Entwickler sowie der Grösse der Community. Da dat grundsätzlich durch die Aufgabenstellung vorgegeben ist, kann dieses Risiko nicht vermieden werden.

3.2 Risikoüberwachung

Die Risiken wurden während des gesamten Projektverlaufs periodisch überwacht und Änderungen entsprechend protokolliert. Idealerweise sollten die Risiken über eine gewisse Zeit hinweg sinken.

3.2.1 1. März 2015

Wie sich herausgestellt hat, ist dat noch weit davon entfernt, produktiv verwendet werden zu können. Ständige API-Änderungen und fehlende Dokumentation machen es zu einem untragbaren Risiko. Im **??** wurde gegen den Einsatz von dat entschieden und somit die Risisken R31 bis R33 eliminiert.

3.2.2 15. April 2015

Nachdem ein erster Prototyp der "Parser" und "Formatter" Konversions-Module sowie der Transformations-Sprache (OpenDataHub Query Language) implementiert wurde, war die grundsätzliche Machbarkeit des Projekts bereits sichergestellt.

3.2.3 15. Mai 2015

Das Risiko R36 (es treten technische Hürden auf, die nicht vorhergesehen wurden und nicht zu bewältigen sind) ist im Zusammenhang der externen Bibliothek GDAL eingetreten. Nach Evaluation diverser Lösungsansätze wurde für die stabilste Version mit grösster Formatunterstützung entschieden. Es ging sehr viel Zeit durch mangelnde/inkonsistente Formatunterstützung dieser Bibliothek verloren.

4 Entwicklungsumgebung und Infrastruktur

In diesem Kapitel wird die Entwicklungsumgebung inkl. der verwendeten Tools beschrieben. **??** zeigt eine Übersicht der Entwicklungsumgebung und den groben Prozess.

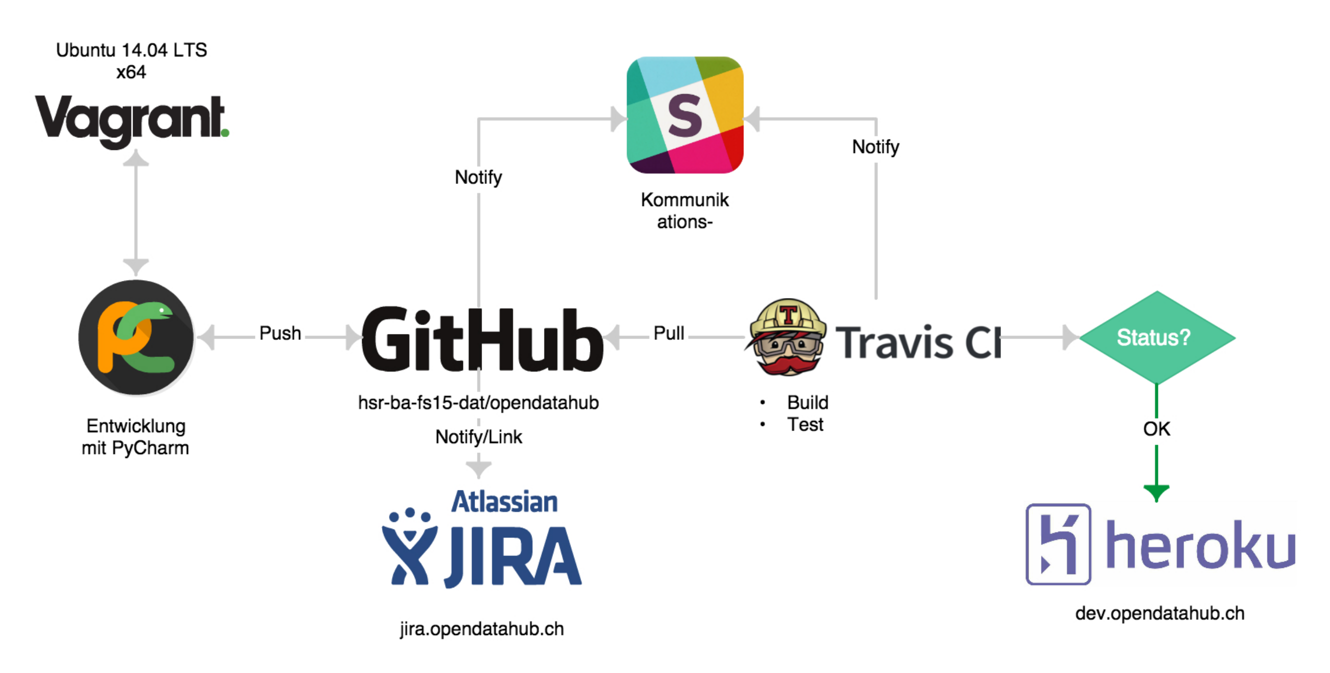


Abbildung 4.1: Entwicklungsumgebung

4.1 IDE

Entscheid 4.1: IDE

Als IDE wurde JetBrains PyCharm verwendet. Sämtliche Mitglieder waren bereits mit dieser IDE vertraut und die neuen Lizenzbestimmungen von JetBrains liessen eine kostenlose Nutzung der professionellen Version zu (Student License).

4.2 SCM

Entscheid 4.2: SCM

Als Source Control Management (SCM) wurden GitHub Repositories innerhalb einer eigenen GitHub Organisation “hsr-ba-fs15-dat” verwendet.

4.3 Projektmanagement

Zur Planung der Ressourcen und Zeit sowie Zuständigkeiten der Tasks wurde das kommerzielle Tool Atlassian Jira verwendet. Jira ist kompatibel mit agilen Vorgehensweisen, insbesondere Scrum, und erlaubt mit einem intuitivem User Interface die Planung von Sprints sowie eine Fortschrittsüberwachung mittels Burndown Charts. **??** zeigt das sogenannte Agile Board im “Work”-Modus während eines Sprints. Die Kopplung von Jira mit GitHub bietet mit der Verwendung von sog. Smart Commits[1] eine noch komfortablere Möglichkeit um Aufwände zu verbuchen oder Tasks zu kommentieren oder zu schliessen.

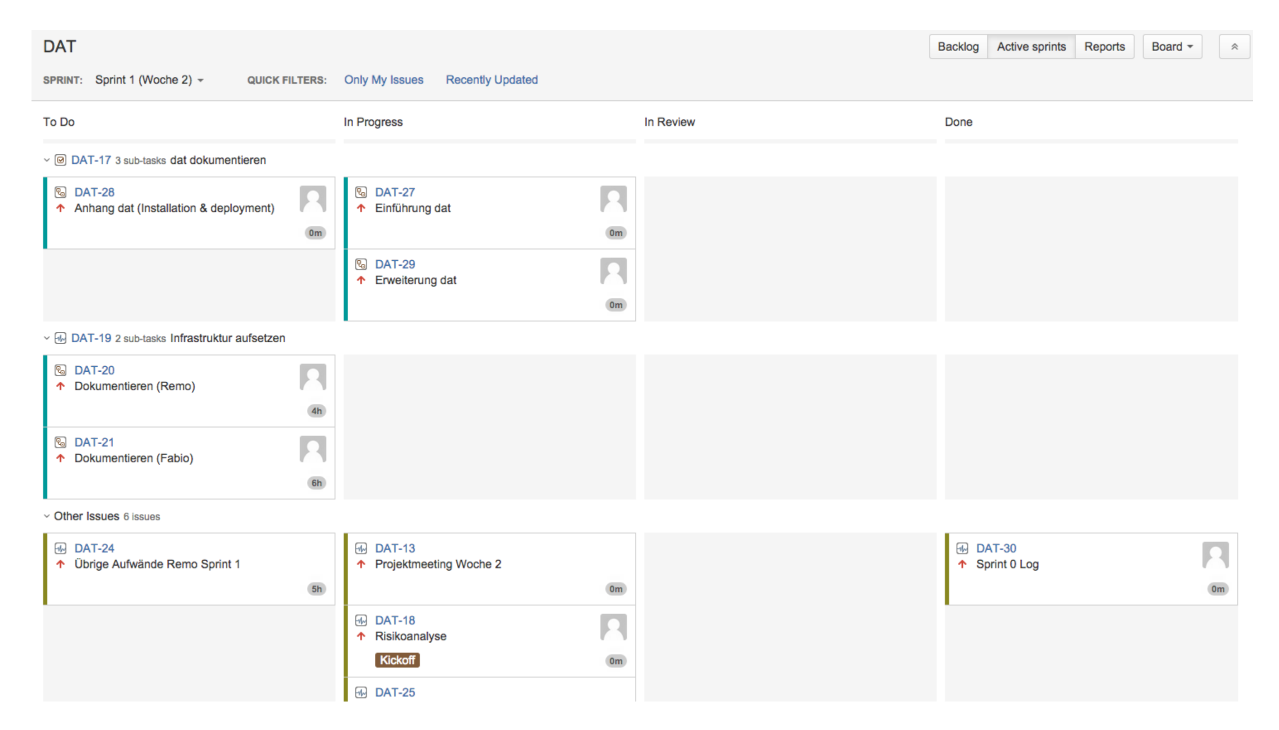


Abbildung 4.2: Jira Agile Board

Jira wurde auf einer virtuellen Maschine der HSR betrieben und dessen Daten täglich automatisiert via cronjob gesichert.

[1] Beispiel: “DAT-18 added project risks #time 3h #comment first draft”
https://confluence.atlassian.com/display/FISHEYE/Using+smart+commits

4.4 Kommunikation

4.4.1 Slack

Bei der Durchführung der Bachelorarbeit wollten wir auf die Kommunikation per E-Mail möglichst verzichten. Das Team hat sich mittels slack.com ausgetauscht sowie Sitzungen protokolliert und Informationen via Weblinks weitergegeben. Slack bietet Team-Kommunikation auf hohem Niveau, mit der Möglichkeit Services wie GitHub, Jira, Travis-CI, E-Mails usw. zu integrieren[2]. So konnte an einem zentralen Ort Bezug auf Commits, E-Mails vom Betreuer und Failing-Builds genommen werden. Durch die Verwendung eines dedizierten Tools zur Kommunikation konnte das E-Mail Postfach von Bachelorarbeit-relevanten Themen freigehalten und direkten Bezug auf konkrete Ereignisse genommen werden. **??** zeigt Screenshots aus dem Mac-Client von Slack.



(a) Git und Travis

(b) Automatische E-Mail Weiterleitung

Abbildung 4.3: Slack Screenshots

4.4.2 Fazit

Die interne Kommunikation wurde vollumfänglich über Slack abgewickelt. Für die Kommunikation mit dem Betreuer und anderen Partnern wurde nach wie vor E-Mail verwendet, da wir Prof. Stefan Keller nicht von dieser Plattform begeistern konnten. Die Vorteile für uns lagen auf der Hand: Wir konnten unsere Mailboxen entlasten und haben unsere gesamte Kommunikation automatisch archiviert.

4.5 Integrationsumgebung

Die Applikation wird nach jedem erfolgreichen Build durch Travis auf Heroku deployed. Die Integrationsinstanz dient zu Demozwecken an Sitzungen und stellt zudem sicher, dass die Webapplikation auch im "kompilierten" Zustand[3] funktioniert.

[2] https://slack.com/integrations

[3] Transpilierung von TypeScript sowie Konkatentierung und "Minification" aller Ressourcen

5 Qualitätsmanagement

5.1 Reviews

Um die Qualität der umgesetzten Tasks zu erhöhen und sicherzustellen wurde ein Review-Prozess eingesetzt. Jeder Task darf erst abgeschlossen werden, wenn ein anderes Teammitglied das Resultat grob angeschaut und für gut befunden hat. Um diesen Prozess einzuhalten wurde ein eigener Jira-Workflow verwendet.

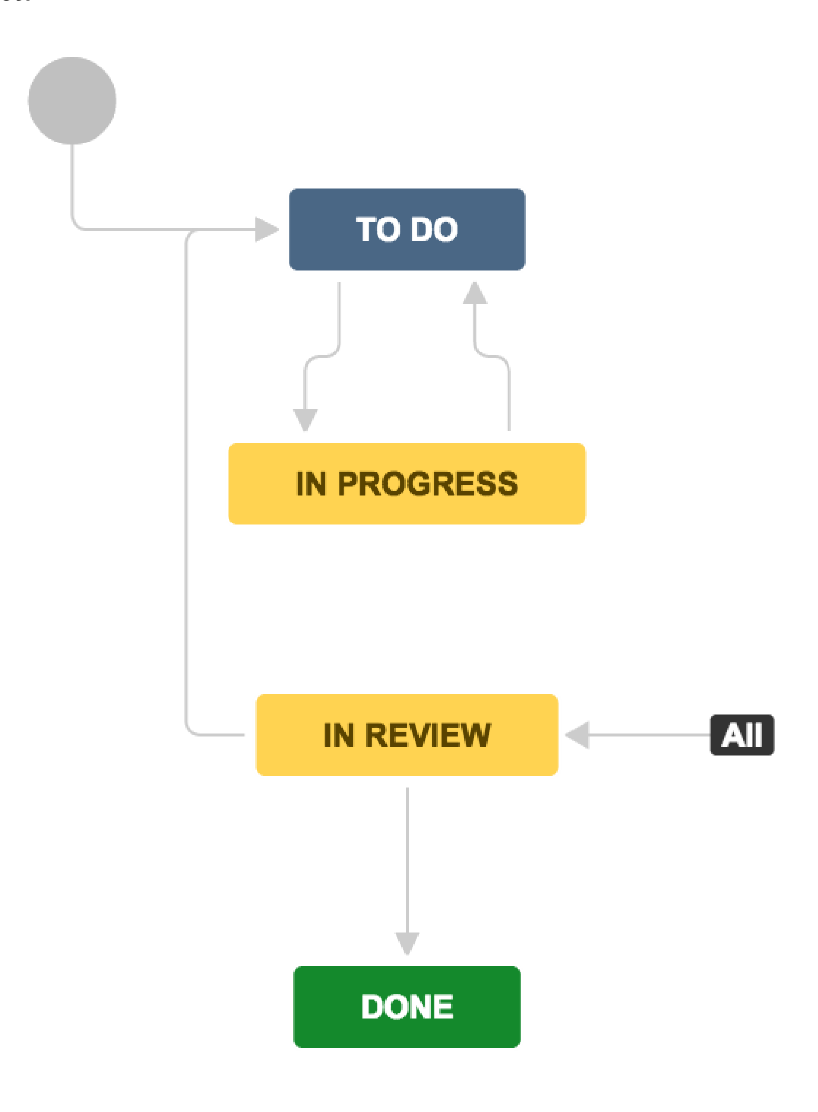


Abbildung 5.1: Jira Review-Workflow

Der in **??** dargestellte Prozess stellt sicher, dass alle Tasks erst in den Review Status versetzt werden

müssen. Von diesem Status aus kann der Task entweder durch den Reviewer geschlossen oder zurück in den Status “To Do” versetzt werden, wobei bei letzterem der Task automatisch dem ursprünglichen Teammitglied zugewiesen wird.

5.2 Programmierrichtlinien

Jeder Entwickler hat seine eigenen Vorstellungen davon, wie Programmcode aussehen soll bzw. welche Richtlinien[1] bei der Entwicklung einzuhalten sind. Aus diesem Grund wurden frühzeitig Programmierrichtlinien inkl. Coding Style Guidelines bestimmt und mittels Entwicklungsumgebung[2] sowie Build-Server forciert.

5.2.1 Python

Für den Python-Quellcode wurden die in der Python Community verbreiteten Python Enhancement Proposal 8 (PEP 8) Richtlinien angewendet. Die einzige Ausnahme stellt hierbei eine Zeilenlänge von 120 statt den nur 80 Zeichen dar. Die PEP 8 Regeln werden teilweise durch die verwendete Entwicklungsumgebung sowie Flake8[3] und PyLint[4] auf dem Continuous Integration (CI) Server streng erzwungen. Bei Verstoss versagt der Build und der Entwickler wird benachrichtigt. Ein Beispiel-Output auf dem Build-Server sieht folgendermassen aus:

```
[INFO]  Executing flake8 on project sources.
[WARN]  flake8: src/main/python/authentication/tests.py:5:9: W292 no newline at end of
 ↪  file
[WARN]  flake8: src/main/python/authentication/tests.py:1:1: F401 'TestCase' imported
 ↪  but unused
[WARN]  flake8: src/main/python/authentication/admin.py:1:1: F401 'admin' imported but
 ↪  unused
[WARN]  flake8: src/main/python/opendatahub/urls.py:8:1: F401 'IndexView' imported but
 ↪  unused
------------------------------------------------------------
BUILD FAILED - flake8 found 4 warning(s)
```

5.2.2 TypeScript

Zur Formatierung des Quellcodes wurden die Standardregeln der Entwicklungsumgebung angewendet. Diese sind unter `opendatahub/idea/settings.jar` abgelegt. Auch hier wurde zur Prüfung und Einhaltung von TypeScript-spezifischen Regeln ein Linter (tslint) verwendet und mittels Build-Server erzwungen. Die Regeln sind in `opendatahub/src/main/webapp/tslint.json` konfiguriert. Da eine Typisierung in TypeScript optional ist, gilt der Grundsatz “dort wo es Sinn macht”, das heisst konkret

[1] Namenskonventionen, Modularität, ...
[2] PyCharm von JetBrains (IntelliJ IDEA)
[3] http://flake8.readthedocs.org
[4] http://www.pylint.org/

- alle Variablen die ein TypeScript Interface besitzen (z.B. AngularJS Services),
- alle public Funktionen und Methoden die auch exportiert werden und somit irgendwo Wiederverwendung finden sowie deren Attribute.

Nicht zu annotieren sind inline Lambda-Funktionen, ausser wenn für deren Parameter ein Interface existiert. Ein Beispiel-Output von tslint sieht folgendermassen aus:

```
Running "tslint:files" (tslint) task
app/scripts/app.config.ts[2, 3]: comment must start with a space
app/scripts/app.config.ts[21, 3]: comment must start with a space
app/scripts/app.config.ts[21, 8]: file should end with a newline
app/scripts/app.ts[37, 11]: comment must start with lowercase letter
app/scripts/app.ts[5, 9]: unused variable: 'openDataHu
app/scripts/auth/controllers/account-settings.controller.ts[30, 15]: comment must
 ↪  start with lowercase letter
app/scripts/auth/controllers/account-settings.controller.ts[34, 19]: comment must
 ↪  start with a space
7 errors in 3 files
Warning: Task "tslint:files" failed. Use --force to continue.
Aborted due to warnings.
------------------------------------------------------------
BUILD FAILED - Errors while running grunt, see
 ↪  /home/travis/build/hsr-ba-fs15-dat/opendatahub/target/reports/grunt.err
```

5.2.3 AngularJS

Bei AngularJS gibt es eine Vielzahl von Möglichkeiten, ein- und dasselbe zu erreichen. Aus diesem Grund ist es umso wichtiger, sich auf bestimmte Vorgehensweisen zu einigen. Grundsätzlich gelten die mittlerweile etablierten und nahezu offiziellen AngularJS Richtlinien und Best Practices von "John Papa"[**angular-styleguide**]. Die Richtlinien sind auf JavaScript ausgelegt. Da wir TypeScript verwenden sind nur diejenigen anzuwenden die auch im TypeScript Kontext zutreffen bzw. noch Sinn machen. Besonders zu beachten sind folgende Regeln:

- Verwendung der "controller as" Syntax um Scope-Vererbung zu vermeiden
- DOM Manipulationen finden ausschliesslich in AngularJS-Direktiven statt
- Single Responsibility Priciple – Eine Klasse/Service/Controller pro Datei mit Ausnahme von kleinen "Sub-Controllern" die zu einem anderen gehören, oder nicht exportierte (private) Klassen.

Es wurden eigene TypeScript "New File" IntelliJ Templates für AngularJS Services, Controller und Direktiven erstellt. Diese stellen sicher, dass jeder Entwickler immer wieder dieselbe Grundstruktur verwendet. Diese sind unter `opendatahub/idea/settings.jar` abgelegt.

6 Sprints

6.1 Sprint 0

Summary

Sprint 0 *(Kickoff)*	
Periode	16.02.2015–22.02.2015
Stunden Soll	48 h
Stunden Plan	–
Stunden Ist	42.25 h

Ziele

- Kickoff
- Aufgabenstellung verstehen
- Infrastruktur aufsetzen

Abgeschlossen

Folgende High-level (ohne Subtasks) Tasks wurden während des ersten Sprints erstellt und sogleich abgeschlossen.

JIRA-Key	Summary
DAT-2	Infrastruktur aufsetzen Task
DAT-3	Dokumentation aufsetzen Task
DAT-4	Projektmeeting Woche 1 (Kickoff)
DAT-15	Grobe Projektplanung
DAT-16	Einlesen dat

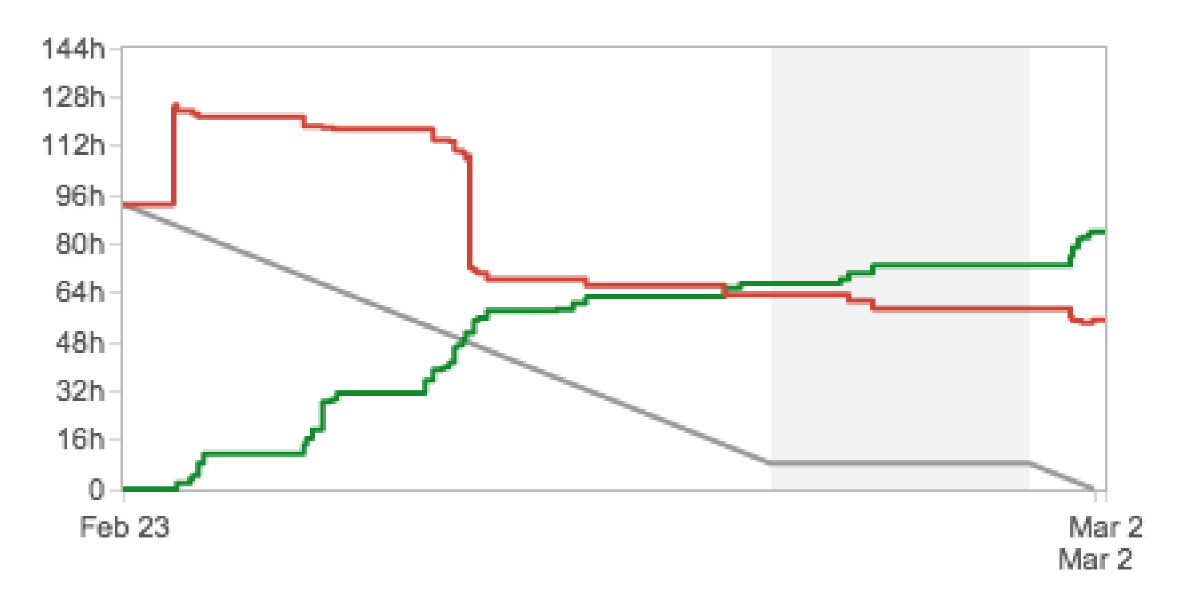
Probleme

Aufgrund einer noch fehlenden Note war die Beteiligung von Christoph Hüsler an diesem Projekt ungewiss. Diese Zweifel konnten rasch geklärt werden und das Team ist froh ihn bei der Bachelorarbeit dabei zu haben.

6.2 Sprint 1

Summary

Sprint 1 *(Einarbeitung dat)*	
Periode	23.02.2015–01.03.2015 12:00 Uhr
Stunden Soll	72 h
Stunden Plan	86.5 h
Stunden Ist	76.5 h



Burndown Chart Sprint 1

Die graue Fläche zeigt den Scope-Change aufgrund der in **??** genannten Gründe.

Ziele

- Dat-Einführung abschliessen
- Risikoanalyse abschliessen
- Basis-Setup Infrastruktur abschliessen

Abgeschlossen

Folgende High-level (ohne Subtasks) Jira Tasks wurden während Sprint 1 abgeschlossen.

JIRA-Key	Summary
DAT-13	Projektmeeting Woche
DAT-17	dat dokumentieren
DAT-18	Risikoanalyse
DAT-19	Infrastruktur aufsetzen
DAT-24	Übrige Aufwände Remo Sprint 1
DAT-25	Übrige Aufwände Fabio Sprint 1
DAT-26	Übrige Aufwände Christoph Sprint 1
DAT-30	Sprint 0 Log

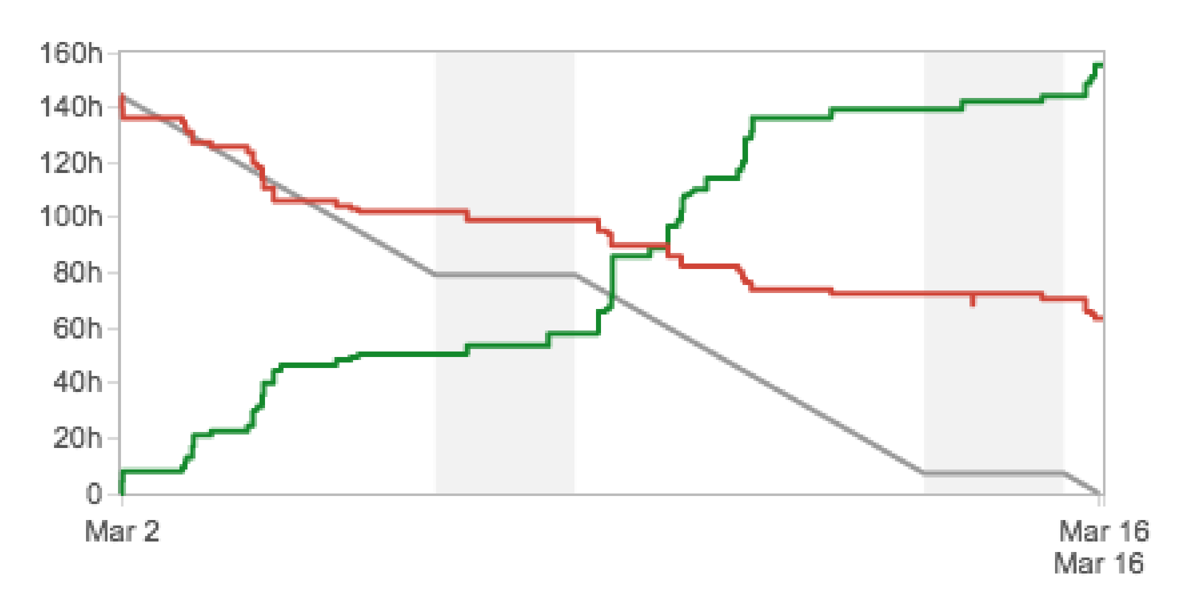
Probleme

Wie sich herausgestellt hat ist dat noch nicht ansatzweise produktionsreif. Aus diesem Grund muss das ursprüngliche Ziel der Arbeit, eine Plattform basierend auf dat zu entwickeln, angepasst werden. Die Grundidee bleibt dieselbe, nur das diese nun ohne dat und somit mit einem komplett anderem Technologiestack umgesetzt wird. Aufgrund dieser Entscheidung wurde der Sprint vorzeitig nach einer Woche beendet und das weitere Vorgehen komplett neu geplant.

6.3 Sprint 2

Summary

Sprint 2	
Periode	02.03.2015 12:00 Uhr–16.03.2015 12:00 Uhr
Stunden Soll	144 h
Stunden Plan	144.3 h
Stunden Ist	157.3 h



Burndown Chart Sprint 2

Ziele

Im Hinblick auf die veränderten Umstände der ursprünglichen Aufgabenstellung muss ein Teil der Infrastruktur neu aufgesetzt werden. Zudem stellt sich die Frage nach der groben Architektur des nun selbst zu implementierenden Extract, Transform, Load (ETL) Prozesses.

- Infrakstruktur basierend auf Python aufsetzen
- Grobe Architekturüberlegungen für ETL
- Use Cases / User Stories konkretisieren
- Erste Human Computer Interaction and Design (HCID) Überlegungen für das Webfrontend
- Einarbeitung Django

Abgeschlossen

Folgende High-level (ohne Subtasks) Jira Tasks wurden während Sprint 2 abgeschlossen.

JIRA-Key	Summary
DAT-14	Projektmeeting Woche 3
DAT-32	Sprint 1 Log
DAT-33	Projektmeeting Woche 4
DAT-34	Organisation, Planung & Kommunikation
DAT-36	Übrige Aufwände Remo Sprint 2
DAT-37	Übrige Aufwände Christoph
DAT-38	Übrige Aufwände Fabio
DAT-40	Epics/User Stories aufnehmen
DAT-41	Architektur Backend
DAT-43	HCID
DAT-44	Django/AngularJS

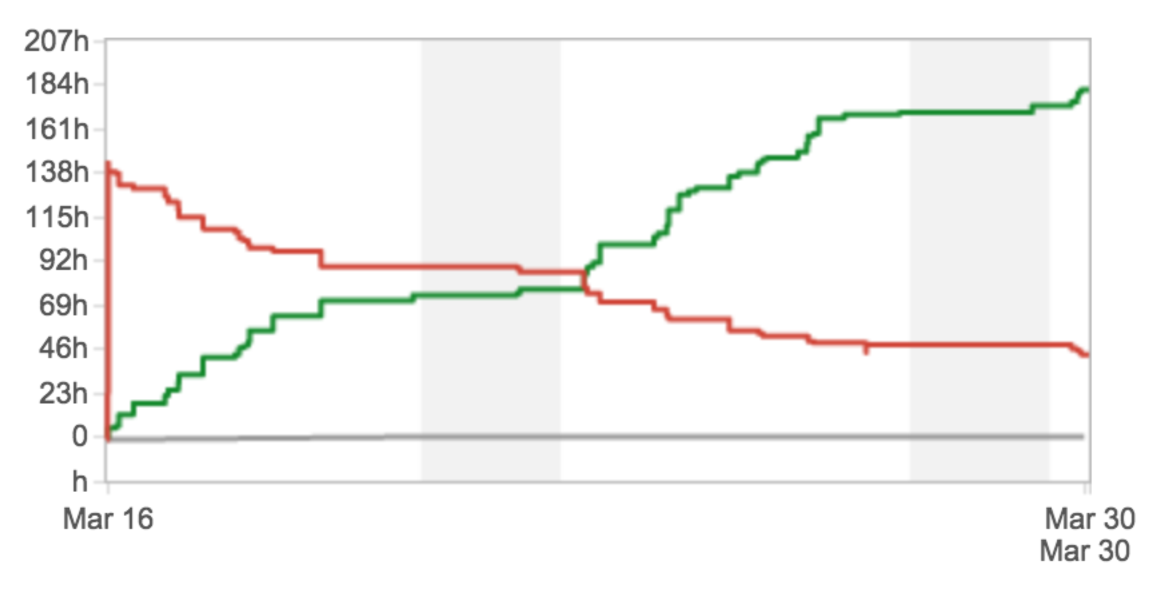
Probleme

Die Einrichtung der Infrastruktur inkl. CI und auto-deployment auf Heroku hat mehr Zeit beansprucht als ursprünglich geplant.

6.4 Sprint 3

Summary

	Sprint 3
Periode	16.03.2015 12:00 Uhr–30.03.2015 12:00 Uhr
Stunden Soll	144 h
Stunden Plan	145 h
Stunden Ist	204.3 h



Burndown Chart Sprint 3

Ziele

Das Hauptziel dieses Sprints ist, ein Fundament für die restliche Applikation/Entwicklung zu schaffen. Es sollten diverse Dateiformate in eine geeignete Datenstruktur geladen werden, um diese wiederum in andere Formate zu konvertieren bzw. "formatieren" zu können.

- "Plugin/Pluggable" Interface für Daten-Extractors/Parsers, Transformers und Formatters
- Diverse Dateiformate einlesen und konvertieren können
- Authentifizierung

Abgeschlossen

Folgende High-level (ohne Subtasks) Jira Tasks wurden während Sprint 3 abgeschlossen. Ab dem nächsten Sprint werden zur besseren Übersicht alle "Übrigen Aufwände" sowie Sitzungen zusammengefasst. Eine Auswertung pro Person bleibt dennoch möglich.

JIRA-Key	Summary
DAT-49	Refactoring Sprint 2
DAT-50	Benutzerverwaltung/OAuth
DAT-51	Liste der Dokumente
DAT-52	Detailansicht Dokument
DAT-56	Besseres Design/Layout
DAT-57	Extractor/Transformer Interface bestimmten
DAT-64	Projektmeeting Woche 5
DAT-65	Projektmeeting Woche 6
DAT-66	Organisation, Planung & Kommunikation
DAT-69	Übrige Aufwände Remo
DAT-70	Übrige Aufwände Fabio
DAT-71	Übrige Aufwände Christoph

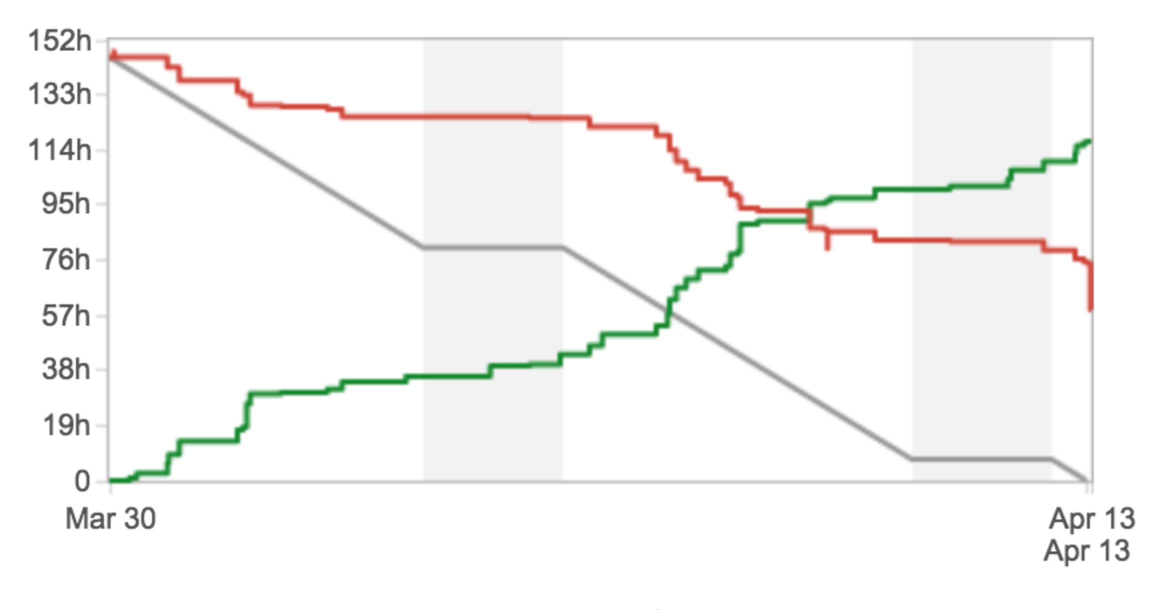
Probleme

Die Authentifizierung via OAuth hat aufgrund erstmaliger Einarbeitung eines Teammitglieds in AngularJS mehr Zeit beansprucht als geplant.

6.5 Sprint 4

Summary

Sprint 4	
Periode	30.03.2015 12:00 Uhr–13.04.2015 12:00 Uhr
Stunden Soll	144 h
Stunden Plan	134 h
Stunden Ist	134.5 h



Burndown Chart Sprint 4

Ziele

Das Hauptziel dieses Sprints liegt in der Entwicklung der Schema-Transformationssprache. Der Favorit zu Beginn des Sprints war die Verwendung eines eigenen SQL Dialekts. Die Vor- und Nachteile verschiedener Lösungen wurden evaluiert.

Abgeschlossen

Folgende High-level (ohne Subtasks) Jira Tasks wurden während Sprint 4 abgeschlossen.

JIRA-Key	Summary
DAT-74	Übrige Aufwände Sprint 4
DAT-75	Organisation, Planung & Kommunikation Sprint 4
DAT-76	Projektmeetings Sprint 4
DAT-78	Basis Transformationen Story
DAT-79	Transformation erstellen

Ein SQL Subset/Dialekt "ODHQL" mit SQL-Basisfunktionalität (Selektion, Joins, Bedingungen, Funktionen) und einigen String- und Geometriefunktionen wurde implementiert. Weitere Funktionen werden bei Bedarf hinzugefügt. Das User-Interface (DAT-79) ist noch im Anfangsstadium und wird in den nächsten Sprints weiterentwickelt.

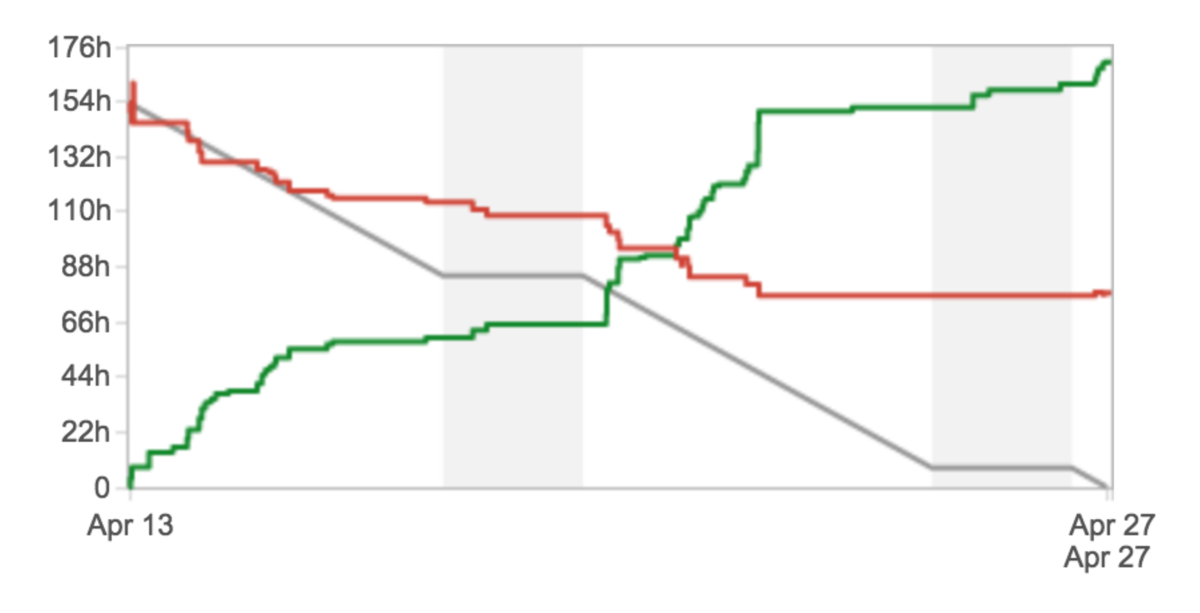
Probleme

Minimale Koordinationsprobleme durch die unterrichtsfreie Zeit über Ostern. Teammitglied Remo Liebi hat viel Zeit durch Ausfall seiner Entwicklungsumgebung und dessen Neuinstallation verloren.

6.6 Sprint 5

Summary

	Sprint 5
Periode	13.04.2015 12:00 Uhr–27.04.2015 12:00 Uhr
Stunden Soll	144 h
Stunden Plan	154 h
Stunden Ist	176.4 h



Burndown Chart Sprint 5

Ziele

In diesem Sprint sollte einerseits die Performance durch Implementation von Caching Mechanismen gesteigert werden. Andererseits sollten Online Datenquellen automatisch in einem konfigurierbaren Intervall aktualisiert werden.

Abgeschlossen

Folgende High-level (ohne Subtasks) Jira Tasks wurden während Sprint 5 abgeschlossen.

JIRA-Key	Summary
DAT-88	Online-Daten auto-refresh
DAT-94	Interlis Kompatibilität
DAT-95	CRUD Transformationen
DAT-96	Caching
DAT-97	Übrige Aufwände Sprint 5
DAT-98	Organisation, Planung & Kommunikation Sprint 5
DAT-99	Projektmeetings Sprint 5

Neu werden alle eingelesenen (geparsten) Dateien in verschiedenen Caches abgelegt, was die Interaktion im User Interface erheblich verschnellert. Zudem werden WFS-Datenquellen automatisch erkannt und können ebenfalls verwendet werden. Das User Interface zur Erstellung der Transformationen ist nach wie vor in Entwicklung.

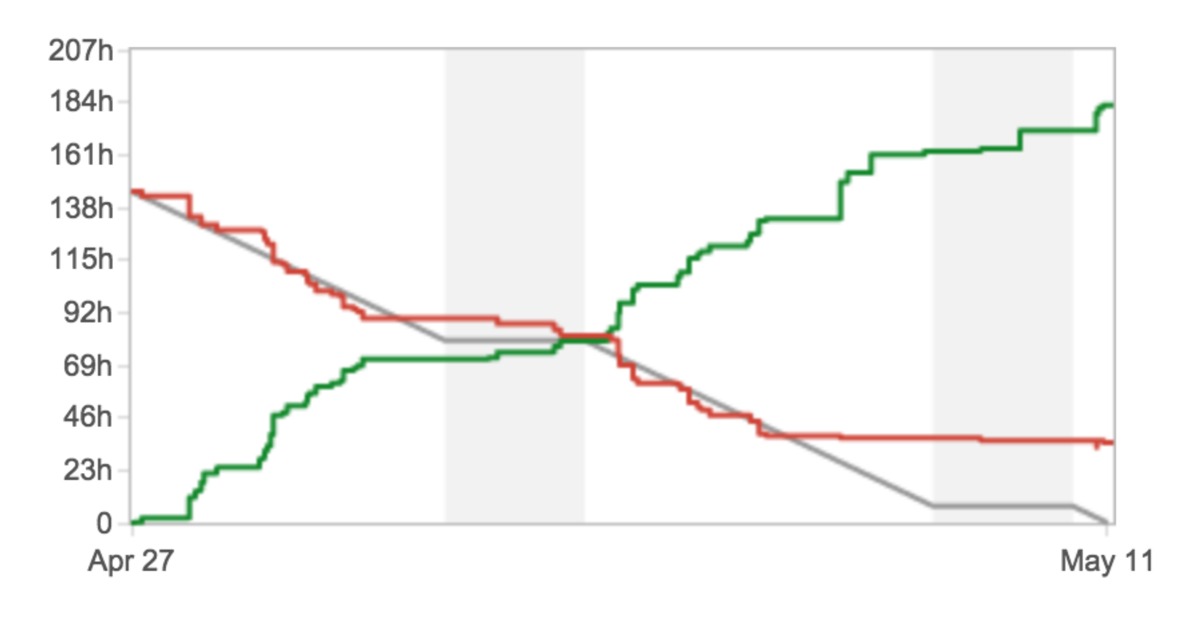
Probleme

Meinungsverschiedenheiten in Team. Diese konnten durch Diskussion beigelegt werden.

6.7 Sprint 6

Summary

Sprint 6	
Periode	27.04.2015 12:00 Uhr–11.05.2015 12:00 Uhr
Stunden Soll	144 h
Stunden Plan	151.5 h
Stunden Ist	167 h



Burndown Chart Sprint 6

Ziele

Dieser Sprint ist der letzte Feature-Sprint. Insbesondere die Transformationen sollten in ihrer Basisfunktionalität fertig implementiert werden. Zudem sollen die diversen Quellen der TROBDB sauber mittels ODHQL-Transformationen integriert werden.

Abgeschlossen

Folgende High-level (ohne Subtasks) Jira Tasks wurden während Sprint 6 abgeschlossen.

JIRA-Key	Summary
DAT-112	Projektmeetings Sprint 6
DAT-113	Organisation, Planung & Kommunikation Sprint 6
DAT-114	Übrige Aufwände Sprint 6
DAT-115	TROBDB Transformation
DAT-116	Template Transformationen
DAT-117	Bugfixing
DAT-119	OdhQL Benutzerdokumentation

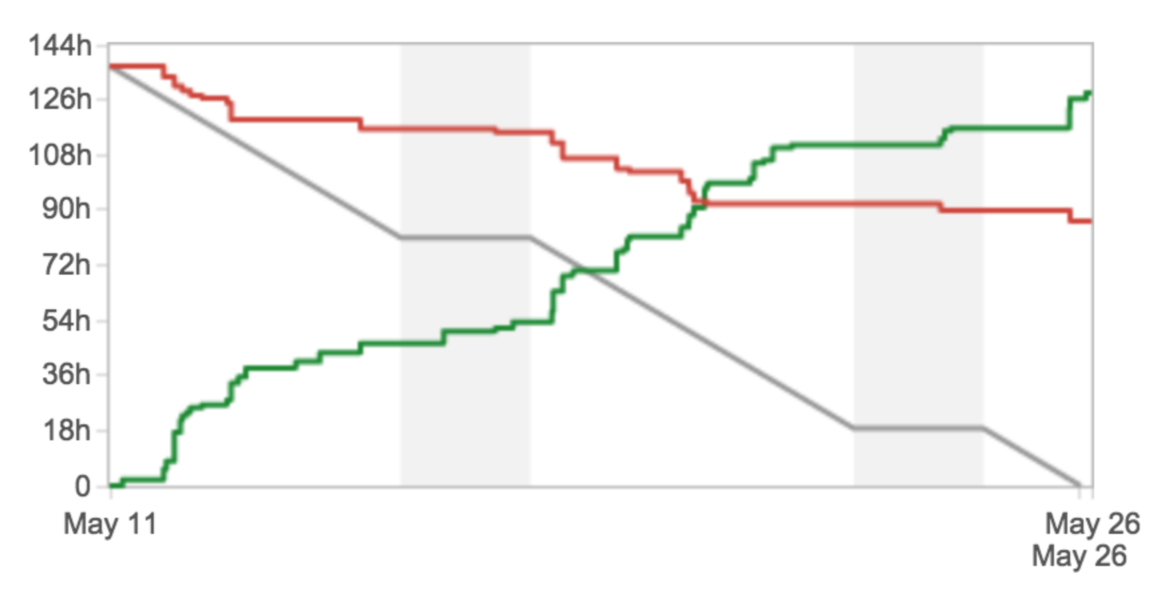
Probleme

Die verschiedenen GDAL-Versionen auf den verschiedenen Systemen (Travis, Heroku, Entwicklung) haben zu unterschiedlichem Verhalten und Fehlern in der Applikation geführt. Neu muss auf allen Systemen die Bibliothek (Version 1.11.1) selbst kompiliert werden (statt via apt-get).

6.8 Sprint 7

Summary

Sprint 7	
Periode	11.05.2015 12:00 Uhr–26.05.2015 12:00 Uhr
Stunden Soll	144 h
Stunden Plan	147 h
Stunden Ist	146.8 h



Burndown Chart Sprint 7

Ziele

Die Kernfunktionalität von OpenDataHub (ODH) steht und die Use-Cases wurden weitestgehend umgesetzt. Somit ist dieser Sprint als reiner Bugfixing oder Improvement/Enhancement Sprint gedacht. Vor allem im Bereich mit Geo-Daten sind noch einige Fehlersituationen denkbar.

Abgeschlossen

Folgende High-level (ohne Subtasks) Jira Tasks wurden während Sprint 7 abgeschlossen.

JIRA-Key	Summary
DAT-139	Bugfixing
DAT-141	UI Tests
DAT-143	Update & Delete Package ("Documents")
DAT-148	Organisation, Planung & Kommunikation Sprint 7
DAT-149	Projektmeetings Sprint 7
DAT-150	Übrige Aufwände Sprint 7
DAT-152	Logo erstellen
DAT-175	Interlis-Modell verwenden für Interlis-Konvertierung
DAT-176	Expressions in Join-Conditions implementieren

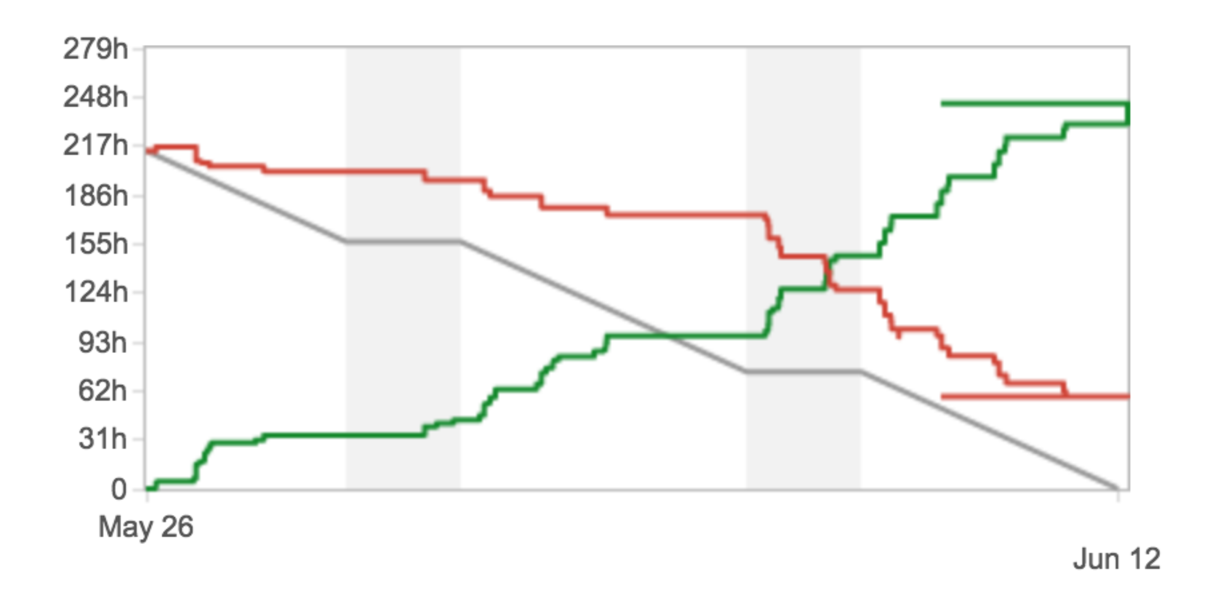
Probleme

Im letzten Sprint wurde aufgrund der fehlenden GeoPackage-Kompatibilität auf GDAL 1.11.1 gewechselt. In diesem Sprint sahen wir uns nach Experimenten mit diversen Versionen inkl. 2.0.0beta1 gezwungen, wieder auf GDAL 1.10.1 zu wechseln.

6.9 Sprint 8

Summary

Sprint 7	
Periode	26.06.2015 12:00 Uhr–12.06.2015 20:00 Uhr
Stunden Soll	216 h
Stunden Plan	224 h
Stunden Ist	255 h



Burndown Chart Sprint 8

Ziele

Dieser letzte, ausnahmsweise dreiwöchige Sprint dient der Finalisierung und Druck der Dokumentation sowie weitere mit der Bachelorarbeit zusammenhängenden organisatorischen Aufwände. Kleinere Last-Minute Bugfixes sind ebenfalls erlaubt.

Abgeschlossen

Folgende High-level (ohne Subtasks) Jira Tasks wurden während Sprint 8 abgeschlossen.

JIRA-Key	Summary
DAT-181	Dokumentation
DAT-202	Übrige Aufwände
DAT-203	Projektmeeting
DAT-204	Last Minute Bugfixes
DAT-207	Organisation, Planung & Kommunikation Sprint 8

Probleme

Keine

6.10 Total und Fazit

Total	
Periode	16.02.2015–12.06.2015
Stunden Soll	960 h
Stunden Ist	1404.3 h

Total pro Person	
Remo Liebi	491 h
Christoph Hüsler	436.5 h
Fabio Scala	476.8 h

Wie in den Burndown Charts und Zeitauswertung der jeweiligen Sprints ersichtlich ist, wurde der Aufwand relativ gut geschätzt. Weniger gut waren die Schätzungen einzelner Tasks, was in den Charts durch die Differenz zwischen verbleibendem Aufwand und verbuchten Aufwand dargestellt wird.

Teil IV

Softwaredokumentation

1 Entwicklung

1.1 Initialisieren

Dieses Kapitel beschreibt das Aufsetzen der Entwicklungsumgebung für ODH. Voraussetzung ist eine Installation von Vagrant.
Vagrant kann für alle gängigen Betriebssysteme von vagrantup.com heruntergeladen werden.

1.1.1 Klonen des Git-Repository

Der nächste Schritt ist das Klonen des Git Repositories und starten der Vagrant VM.

```
git clone git@github.com:hsr-ba-fs15-dat/vm.git  # lädt das Repository von github.com
cd vm  # wechselt in das gerade heruntergeladene Verzeichnis
vagrant up  # Startet die Vagrant-Instanz
```

Wenn diese Befehle in eine *nix Konsole eingegeben werden, wird automatisch eine passende VM heruntergeladen und konfiguriert. Unter Umständen muss die Konfiguration angepasst werden. Dies kann im Config-File von puppet gemacht werden. (vm/puphpet/config.yaml)

1.1.2 Initialisieren von Django

Nach folgenden Befehlen kann mit der Entwicklung gestartet werden:

```
vagrant ssh # Verbindet mit der SSH Umgebung
make dev # Initialisiert die venv und installiert weitere benötigte Packete
exit # Ausloggen ist notwendig, damit das virtuelle environment (virtualenv) korrekt
 ↪  geladen wird.
```

1.2 Entwicklungsumgebung

1.2.1 PyCharm

Wir nutzten zur Entwicklung von OpenDataHub die Entwicklungsumgebung PyCharm für Python aus dem Hause JetBrains. Für Studenten ist die Professional Edition kostenlos verfügbar.

1.2.2 Projekt erstellen

Nun kann ein PyCharm-Projekt erstellt werden. Dazu wählt man "Datei → öffnen" und wählt den Ordner `opendatahub` aus. Das Projekt wird erstellt.

1.2.3 Python-Interpreter

???? beschreiben die Einstellungen, die vorgenommen werden müssen, damit alle Debugging Features funktionieren.

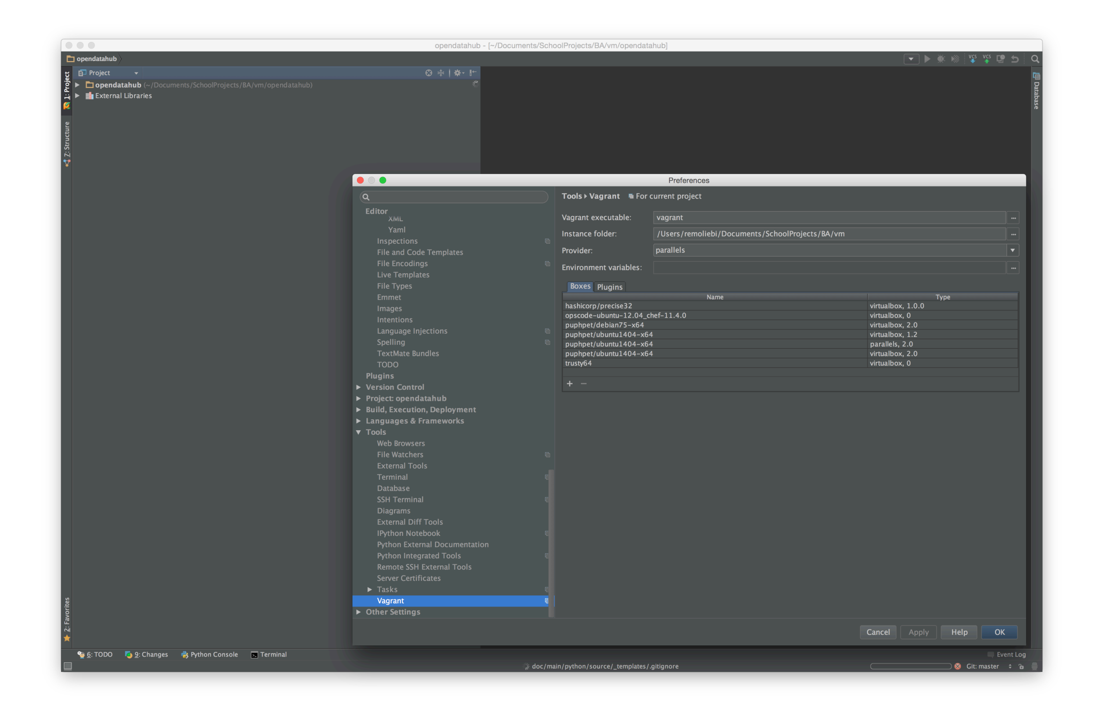


Abbildung 1.1: Übersicht Einstellungen: Vagrant

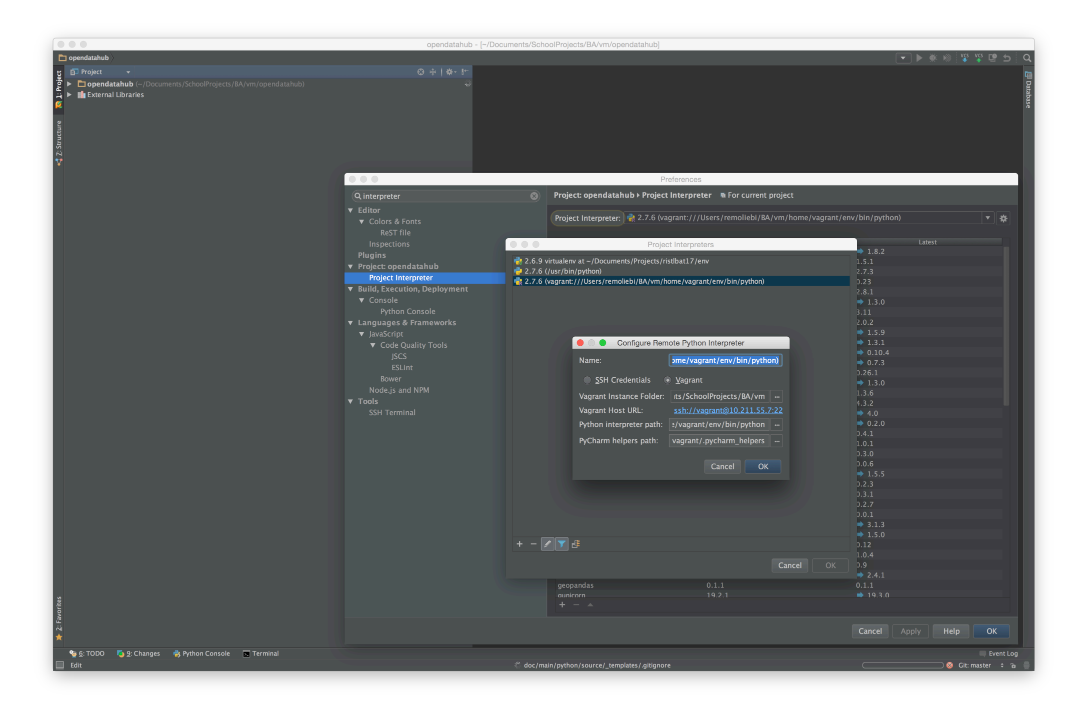


Abbildung 1.2: Übersicht Einstellungen: Projektinterpreter

1.2.4 File-Watchers

Für die Verwendung von TypeScript sollte ein File-Watcher in PyCharm eingerichtet werden. Dies wird in **??** beschrieben.

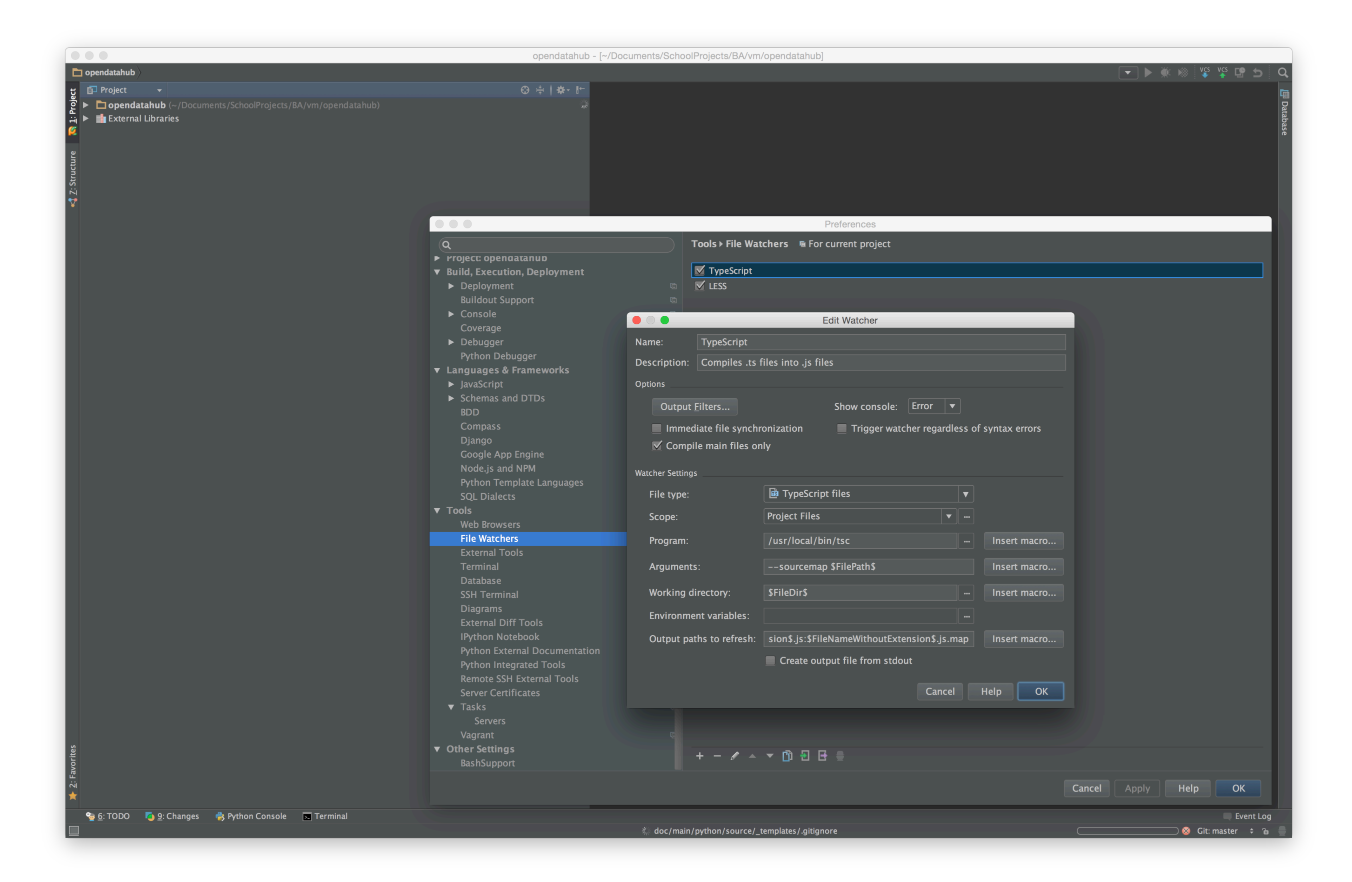


Abbildung 1.3: Übersicht Einstellungen: FileWatcher: TypeScript

Für die Verwendung von TypeScript muss das Paket "tsc" auf der lokalen Maschine installiert sein.

1.2.5 Build

Das ganze Projekt muss in der virtuellen Maschine (VM) gebuildet werden.

```
cd <PROJECT SOURCE>/vm
vagrant ssh #verbindet direkt in die vm
pyb -v #startet den Build-Prozess
```

1.2.6 grunt

Für die Entwicklung im FrontEnd kann ein kürzerer Weg gewählt werden –Grunt. Grunt wird wie folgt aufgeführt

```
vagrant ssh #verbindet direkt in die vm
cd src/main/webapp # in den FrontEnd Ordner wechseln
grunt # FrontEnd Build startet.
```

Dabei werden sämtliche Less- und TS-Files kompiliert und geprüft.

1.2.7 Deployment

Seit der Vagrant Version 1.7 vom Dezember 2014 ist Deployment mittels Vagrant möglich. Durch den Befehl `vagrant push` kann, je nach Konfiguration, auf Heroku, SFTP und FTP sowie durch selbstgeschriebene Kommandozeilenskripte oder Atlas deployed werden.
Syntaxbeispiel für einen FTP-push: [**vagrant-deployment**]

```
config.push.define "ftp" do |push|
  push.host = "ftp.test.com"
  push.username = "benutzer1"
  push.password = "Passwort1"
  push.secure = false
  push.destination = "/"
  push.dir = "/"
end
```

Diese Konfiguration ist in unserem Vagrant File nicht enthalten und muss je nach FTP-Server oder Heroku-Instanz konfiguriert werden.

1.2.8 Tests

Beim Build-Prozess werden Tests automatisch ausgeführt. Um nur die Python-Tests auszuführen wird folgender Befehl verwendet:

```
pyb -v django_test
```

1.3 Einstellungen

In diesem Kapitel werden die wichtigsten Einstellungen für den Betrieb von OpenDataHub erläutert.

1.3.1 settings.py

Folgende Parameter sollten direkt in der Datei "settings.py" konfiguriert werden:

SECRET_KEY Sollte auf jeder Instanz anders sein.

USE_X_FORWARDED_HOST Verwaltet das Protokoll hinter einem Reverse Proxy

SECURE_PROXY_SSL_HEADER Verwaltet das Protokoll hinter einem Reverse Proxy

DATABASES konfiguriert die Entwicklungs-Datenbank. Hier sollte keine Produktive Datenbank hinterlegt werden. (Siehe **??**)

CACHES konfiguriert die Cache Backend.

LANGUAGE_CODE [de-ch] setzt den I18N Code für Django

TIME_ZONE [Europe/Zurich]

EMAIL_* Für Fehler Benachrichtigungen im Frontend.

PACKAGE_PREFIX [ODH] Prefix für Dokumente.

TRANSFORMATION_PREFIX [TRF] Prefix für Transformationen.

Für diese Parameter gilt, dass sie nicht auf Produktionsumgebungen geändert werden sollten!

1.3.2 Umgebungsvariablen

Umgebungsvariablen müssen dem Programm von aussen mitgegeben werden. Dies hängt von der jeweiligen Art des Umgebung ab. Für PyCharm finden Sie diese Einstellungen unter "Run/Debug Configuration".

DJANGO_DEBUG [False] Wird verwendet um DEBUG Output zu aktivieren.

DJANGO_CONFIG [None] bewirkt, dass die "minified" JavaScript Libraries geladen werden. Dazu auf "PRODUCTION" setzen.

DATABASE_URL definiert die Datenbank URL.
Format: `postgres://benutzer:password@localhost:5432/datenbank`

GITHUB_SECRET Definiert den GitHub Secret Key für OAuth2

GITHUB_PUBLIC Definiert den GitHub Public Key für OAuth2

FACEBOOK_SECRET Definiert den Facebook Secret Key für OAuth2

FACEBOOK_PUBLIC Definiert den Facebook Public Key für OAuth2

DJANGO_SETTINGS_MODULE [opendatahub.settings] es kann eine alternative Einstellungs-Datei geladen werden.

PATH [/vagrant/opendatahub/interlis] Pfad zum Interlis Compiler

CI [False] Reduziert die Tests für Continuous Integration Plattformen. (Nicht lokal einsetzen.)

1.4 Verwendete Software

In diesem Abschnitt wird die in OpenDataHub verwendete Software aufgelistet.

1.4.1 Python

Es wird Python 2.7 mit den in **??** gelisteten Modulen benötigt, um OpenDataHub zu verwenden. Diese sind auch in der Datei `requirements.txt` im Quellcode-Verzeichnis gelistet. Zusätzliche, nur zur Entwicklung benötigte Module sind in **??** sowie in `requirements_dev.txt` gelistet.

Name	Version	Beschreibung
Django		
django-toolbelt	0.0.1	Installiert einige der untenstehenden Packages (gunicorn, dj-static, ...) automatisch.
dj-database-url	0.3.0	Helper um Datenbank-URL aus der Umgebungsvariable zu lesen
Django	1.7.5	Django Framework
dj-static	0.0.6	Static-Asset Server für Gunicorn
gunicorn	19.2.1	WSGI Server
psycopg2	2.6	PostgreSQL Adapter
static3	0.5.1	Static-Asset Server. Durch django-toolbelt installiert.
requests	2.5.3	HTTP Bibliothek
python-social-auth		OAuth Unterstützung
django-picklefield	0.3.1	Erlaubt Python Pickles als Datenbank-Felder
djangorestframework	3.0.5	REST API Framework
django-pgjson	0.2.3	JSON Datentyp-Unterstützung für PostgreSQL
djangorestframework-jwt	1.4.0	JWT-Unterstützung für REST das Framework
whitenoise	1.0.6	Static-Asset Server. Dieser wird effektiv in OpenDataHub verwendet.
NumPy / Pandas		
numpy	1.9.1	High-performance Vektor-Operationen
pandas	0.16.0	Data-Analysis Bibliothek
xlsxwriter	0.7.1	OpenXML Excel (xlsx) Unterstützung für Pandas
xlwt	0.7.5	Excel Unterstützung (schreibend) für Pandas
xlrd	0.9.3	Excel Unterstützung (lesend) für Pandas
Geo		
pygdal	1.10.1	GDAL Python API
geopandas	0.1.1	Pandas Erweiterung um Geo-Funktionalität
fiona	1.5.1	Wrapper um GDAL API, wird von GeoPandas verwendet.
shapely	1.5.7	Geometrie-Objekte in Python. Wird von GeoPandas verwendet.
pyproj	1.9.4	PROJ4 Python API (Koordinaten-Transformationen)
fastkml	0.9	KML Parser Bibliothek

Name	Version	Beschreibung
Diverses		
unicodecsv	0.9.4	Unicode-Unterstützung für CSV.
pyparsing	2.0.3	Parser Parser Bibliothek. Wird verwendet um die OpenDataHub Query Language (ODHQL) zu definieren und parsen.
lxml	3.4.2	LXML Python API
beautifulsoup4	4.3.2	HTML Parser
django-sslify	0.2.0	SSL Unterstützung für Django
enum34	1.0.4	Python 3 Enum-Backport
defusedxml	0.4.1	Security-Wrapper für diverse XML Bibliotheken
Pygments	2.0.2	Syntax Highlighting
docutils	0.12	Dokumentations-Bibliothek. Wird für die ODHQL-API verwendet.
Indirekte Dependencies		
html5lib	0.999	–
click	3.3	–
cligj	0.1.0	–
descartes	1.0.1	–
matplotlib	1.4.3	–
mock	1.0.1	–
python-dateutil	2.4.1	–
pytz	2015.2	–
pillow	2.8.1	–
pyjwt	1.0.0	–

Tabelle 1.1: Zur Laufzeit benötigte Python Module

Name	Version	Beschreibung
sphinx	1.2.3	Generierung der API Dokumentation aus Inline Kommentaren
nose	1.3.4	Test-Runner
pybuilder	0.10.51	Build-tool (Task-runner) für Python
sphinxcontrib-httpdomain	1.3.0	HTTP Erweiterung für Sphinx zur Dokumentation von APIs
django-extensions	1.5.0	Django Erweiterungen/Befehle zur Entwicklung

Name	Version	Beschreibung
werkzeug	0.10.1	Django Middlerware zur Entwicklung
pyinotify	0.9.5	Auto-reload Unterstützung für Django Debug-Server (via inotify)
watchdog	0.8.3	Auto-reload Unterstützung für Django Debug-Server
pylint-django	0.6	PyLint Unterstützung für Django-spezifische Module
ipython	latest	Bessere Debugging-Konsole

Tabelle 1.2: Zur Entwicklung benötigte Python Module

1.4.2 JavaScript

Im Frontend werden die in **??** gelisteten JavaScript Module eingesetzt. Diese sind zudem in `src/main/webapp/bower.json` abgelegt und werden mittels Bower[1] installiert.

Name	Version	Beschreibung
Angular / Core		
angular	1.4.0	AngularJS, Singe-Page-Application Framework
bootstrap	3.2.0	CSS Layout Framework
angular-animate	1.4.0	CSS Animationen mit AngularJS
angular-aria	1.4.0	Accessibility für AngularJS Applikationen
angular-cookies	1.4.0	AngularJS Coookie Untersützung
angular-messages	1.4.0	AngularJS Validierungs- und Fehlermeldungen
angular-route	1.4.0	URL-Routing
angular-sanitize	1.4.0	Security (HTML Escaping) in der Templates
angular-touch	1.4.0	Unterstützung für Touch-Events
json3	3.2.6	Cross-Browser bzw. JavaScript Engine JSON Implementation
es5-shim	2.1.0	Cross-Browser ECMAScript5 Compliance (Shim)
jquery	2.1.3	jQuery
Erweiterungen		
angular-ui-router	0.2.13	Erweitertes URL-Routing
animate.css	3.2.1	Auswahl an CSS3 Animationen

[1] Web Package Manager, siehe `http://bower.io/`

Name	Version	Beschreibung
ngtoast	1.5.0	Anzeige von “Toasts” (Meldungen)
angular-bootstrap	0.12.1	Integration von Bootstrap Elementen in AngularJS
angular-ui-utils	0.2.2	Diverse Common Utilities
angular-ui-select	0.11.1	Erweitertes Selection/Drop-down für AngularJS
font-awesome	4.3.0	Font Awesome Icons
ng-file-upload	3.2.4	File-Upload Direktive für AngularJS
bootstrap-material-design	0.3.0	Google Material Design für Bootstrap
arrive	2.1.0	DOM Watching. Von material-design benötigt zur Integration mit AngularJS.
angular-easyfb	1.2.1	Facebook OAuth Authentication
restangular	1.4.0	REST Client/Bibliothek
bootstrap-social-buttons	1.0.0	Diverse Social Network Icons/Buttons
satellizer	Eigener Fork	Token-Based Authentication
blockui	latest	Utility um einzelne Elemente zu blockieren
angular-truncate	latest	AngularJS Filter
ng-table	0.5.4	AngularJS Tabellen-Erweiterung um Funktionen wie Pagination, Sortierung, . . . ,
angular-ui-ace	latest	Editor/Textfeld mit Syntax-Highlighting
fontawesome-actions	0.5.0	Kombination mehreren Font Awesome Icons
angular-moment	0.10.0	Darstellung von relativen Zeiten
stacktrace-js	0.6.4	Cross-Browser Stacktraces (Error handling)
ES6StringFormat	latest	Backport von `String.format` aus ECMAScript6
angular-scroll	0.7.0	Anchor-Unterstützung für AngularJS
highlightjs	8.5.0	Syntax Highlighting
angular-highlightjs	0.4.1	AngularJS Direktive für highlightjs

Tabelle 1.3: Benötigte JavaScript Module

2 Benutzerhandbuch

2.1 Datensuche und Bezug

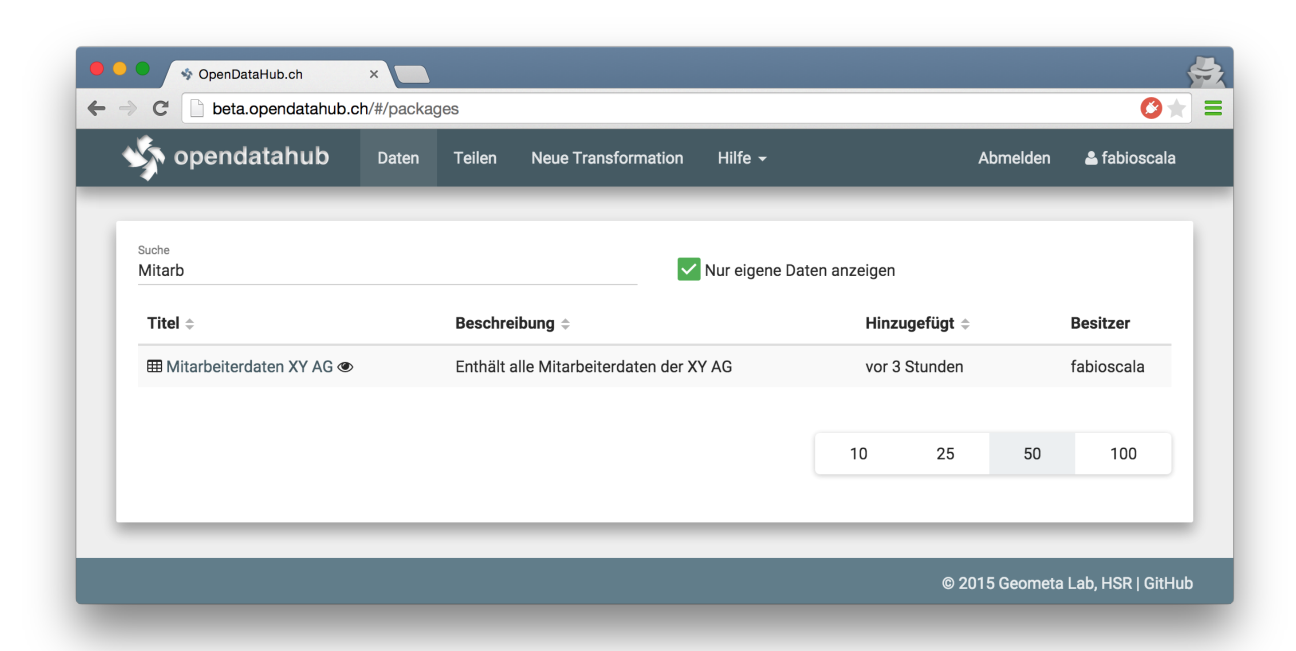


Abbildung 2.1: Suche von Daten in OpenDataHub

1. beta.opendatahub.ch → Daten aufrufen. Es wird eine Liste mit allen in OpenDataHub verfügbaren Daten präsentiert.
   - Hierbei handelt es sich um sogenannte "Packages". Ein Package enthält potenziell mehrere Daten/Tabellen. Auch Transformationen von Daten oder gar Transformationen anderer Transformationen werden über diese zentrale Stelle gelistet.
2. Mittels dem Suchfeld kann im hinterlegten Titel sowie Beschreibung der Daten und Transformationen gefiltert werden. Optional können auch nur die selbst erstellten Daten angezeigt werden.
3. Durch Klick auf den Titel werden Detailinformationen angezeigt. Diese Ansicht wird in den folgenden Abschnitten erläutert.

2.1.1 Daten herunterladen

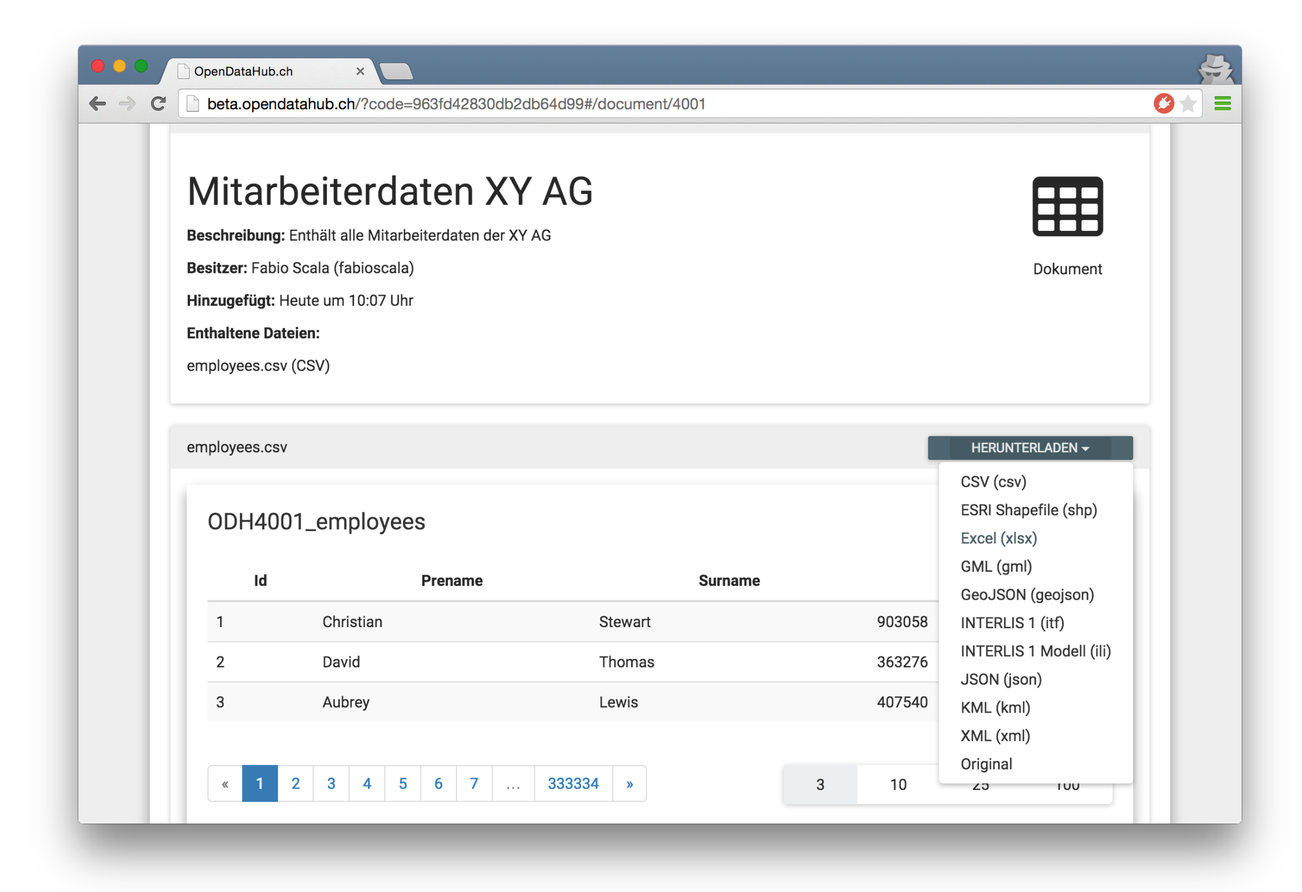


Abbildung 2.2: Herunterladen von Daten

1. Durch Klick auf "Herunterladen" erscheint eine Liste der möglichen Bezugsformate wie in **??**.

2. Es kann nun ein Format wie beispielsweise Excel angeklickt werden.

3. Die Applikation bzw. der Browser wird nun eine Datei zum Download anbieten.

2.1.2 Daten bearbeiten/löschen

Hochgeladene Daten sowie erstellte Transformationen können jeweils gelöscht und bis zu einem gewissen Grad wie folgt bearbeitet werden:

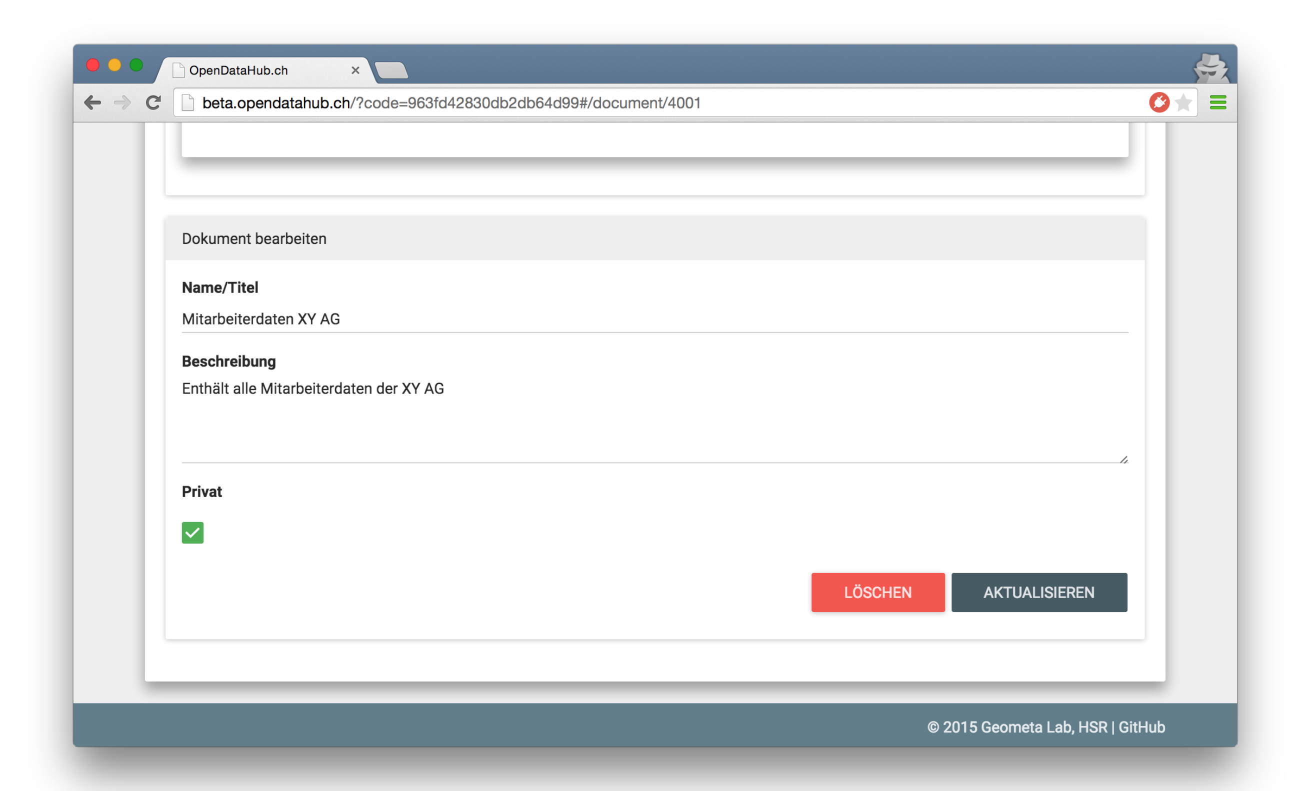


Abbildung 2.3: Bearbeiten von Daten

1. Nachdem die gewünschten Daten wie in **??** beschrieben gefunden wurden, können diese, sofern man der Besitzer/Ersteller ist, bearbeitet werden.

2. Zuunterst in der Detailansicht können die Daten modifiziert oder gelöscht werden.

2.2 Registrierung/Anmeldung

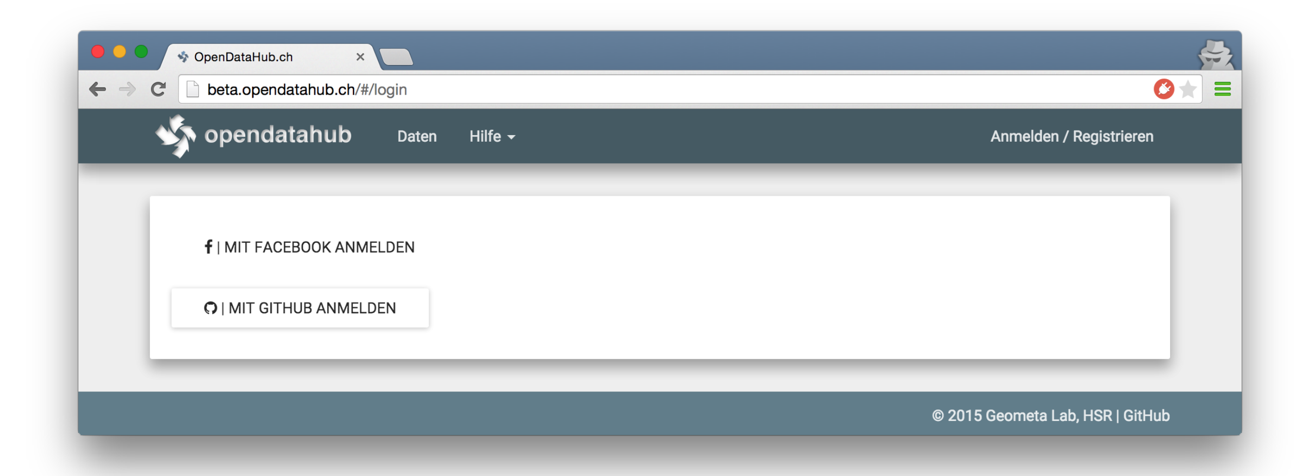


Abbildung 2.4: Anmeldung/Registrierung auf OpenDataHub

Der Bezug von öffentlichen Daten auf OpenDataHub ist stets auch als anonymer Benutzer möglich. Um jedoch eigene Daten zur Verfügung zu stellen oder existierende Daten zu transformieren, ist es nötig sich gegenüber der Applikation zu identifizieren. Dies ist unter beta.opendatahub.ch → Anmelden möglich und setzt ein Benutzerkonto bei einem der folgenden Anbieter voraus:

facebook.com – Soziales Netzwerk. Viele Personen besitzen bereits ein Konto bei Facebook.

github.com – Online Git Repository Hosting. Die meisten Entwickler besitzen bereits ein GitHub-Konto.

2.3 Daten teilen

In diesem Abschnitt wird erklärt, welche Möglichkeiten OpenDataHub bietet um Daten bereitzustellen.

Wichtig Zur Bereitstellung von Daten ist eine Anmeldung erforderlich. Diese ist in **??** beschrieben.

2.3.1 Lokale Daten hochladen

Die erste Möglichkeit besteht darin, Dateien die lokal auf dem PC gespeichert sind, einmalig auf OpenDataHub hochzuladen.

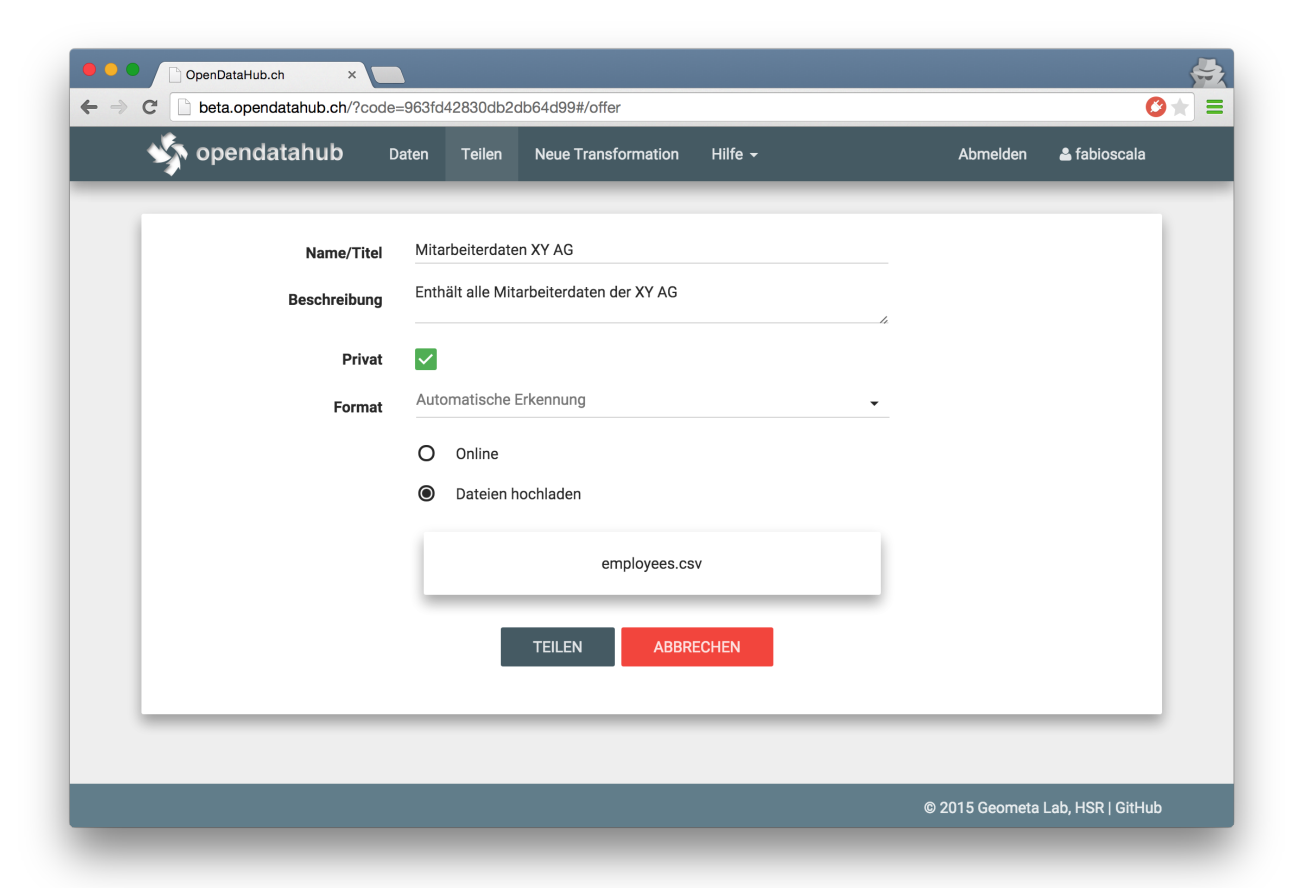


Abbildung 2.5: Dateien in OpenDataHub teilen

?? zeigt den nachfolgend beschriebenen Prozess.

1. beta.opendatahub.ch → Teilen aufrufen.
2. Kurzen Namen/Bezeichnung für die Daten angeben.
3. Beschreibungstext erfassen, der die Daten und deren Inhalt näher beschreibt.
4. Die Checkbox "Privat" gibt an, ob die Daten nur vom angemeldeten Benutzter gesehen werden sollen.
   - Dies ist dann nützlich, wenn der Benutzer OpenDataHub nur als Transformationsplattform nutzen will ohne die Daten freizugeben.
   - Ein zweiter Anwendungsfall wäre ein Shared Account für eine Organisation, um die Daten nur innerhalb dieser Organisation freizugeben.
5. "Dateien hochladen" selektieren.
6. Falls man nur Daten mit dem gleichen Format hochlädt und diese nicht durch deren Dateiendung oder Inhalt erkannt werden, kann Optional das Format der Daten explizit selektiert werden. Es empfiehlt sich dieses jedoch standardmässig auf "Automatisch" zu belassen.

7. Dateien entweder via Drag & Drop direkt ins rechteckige Feld mit der Bezeichnung "Wählen oder ziehen Sie Ihre Dateien" ziehen oder nach Klick auf das Feld über den Datei-Explorer auswählen.
    - Es ist erlaubt, mehrere Dateien (beispielsweise eine CSV Datei und eine XML Datei) hochzuladen.
    - Einige Formate wie beispielsweise das ESRI Shapefile erfordern sogar zwingend mehrere Dateien. Diese müssen vor der Endung denselben Namen besitzen.
    - Allgemein werden alle Dateien mit gleichem Namen und lediglich anderer Endung zu einer Gruppe zusammengefasst. Es ist somit nicht möglich ein `Mitarbeiter.xlsx` und `Mitarbeiter.csv` hochzuladen und dann als separate "Tabellen" zu behandeln.
8. Durch Klick auf "Teilen" werden die Dateien hochgeladen und auf OpenDataHub zur Weiterverarbeitung abgelegt.
9. Wenn OpenDataHub die Daten interpretieren konnte, wird zur Detailansicht weitergeleitet. Ansonsten wird eine Fehlermeldung eingeblendet. In diesem Fall wird der Fehler direkt den Entwicklern weitergeleitet. Wenn es sich um ein nicht unterstütztes Format handelt oder man der Meinung ist, dass es hätte funktionieren sollen, kann man die Datei dem Entwicklungsteam an mailto:devs@opendatahub.ch weiterleiten.

2.3.2 Online Daten hinzufügen

Die zweite Möglichkeit, welche sich vor allem für häufig ändernde oder gar Echtzeit-Daten eignet, ist das Hinzufügen mittels einer Webadresse. OpenDataHub kann die Daten dann selbst in einem gewünschten Zeitintervall aktualisieren.
Dies läuft weitestgehend gleich ab wie in **??** ab. Der einzige Unterschied besteht darin, dass "Online Daten" selektiert und dann eine Webadresse hinterlegt wird. Es werden folgende Arten von Online Daten unterstützt:

- Alle herkömmlichen Adressen die auf Dateien verweisen wie beispielsweise `http://www.hsr.ch/meine-test-datei.xml`, dasselbe gilt für `https://`
- WFS Webservice Adressen wie beispielsweise http://maps.zh.ch/wfs/TbaBaustellenZHWFS.

Die Angabe des Abfrage-Intervalls gibt an, nach welcher Zeit die Daten als "veraltet" eingestuft werden und neu von der Online-Quelle angefordert werden. Wichtig dabei ist, dass OpenDataHub die Daten nicht "aktiv" in diesem Intervall abfragt, sondern erst dann, wenn diese in OpenDataHub auch wirklich benötigt werden.

2.4 Daten transformieren

Wichtig Zur Erstellung von Transformationen ist eine Anmeldung erforderlich. Diese ist in **??** beschrieben.

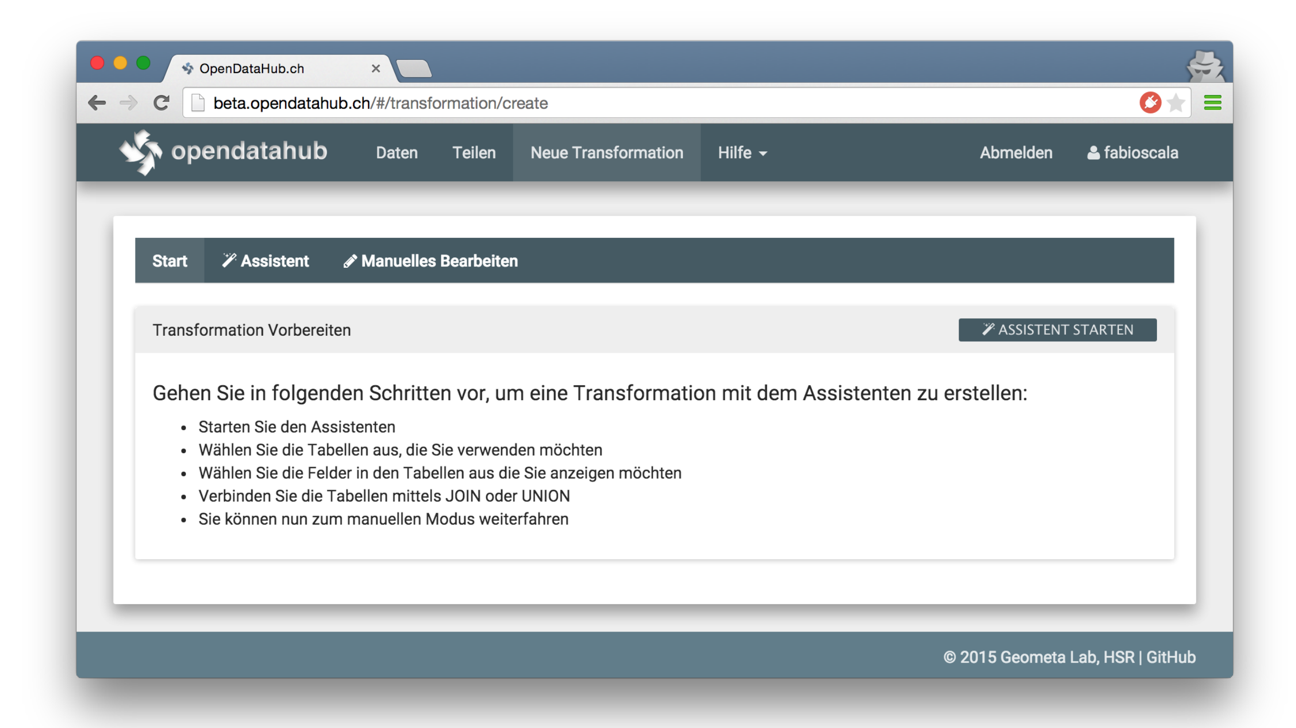


Abbildung 2.6: Startseite für neue Transformation

1. beta.opendatahub.ch → Neue Transformation aufrufen

2. Auf der Seite wie in **??** abgebildet die Instruktionen lesen/beachten

3. Man hat nun folgende zwei Möglichkeiten

   - Der "Assistent" erlaubt mittels einigen Klicks eine simple Transformation bzw. ODHQL

   - Im "Manuelles Bearbeiten" Modus kann direkt mittels ODHQL eine Transformation geschrieben werden.

   In der Praxis empfiehlt sich eine Kombination dieser beiden Varianten: Erst den Assistenten zu verwenden und die entstehende Abfrage dann manuell zu bearbeiten.

2.4.1 Verwendung des Assistenten

Mit dem Assistenten können weniger versierte Benutzer schnell eine Transformation zur Selektion oder Zusammenführung von Daten erstellen.

Datenselektion

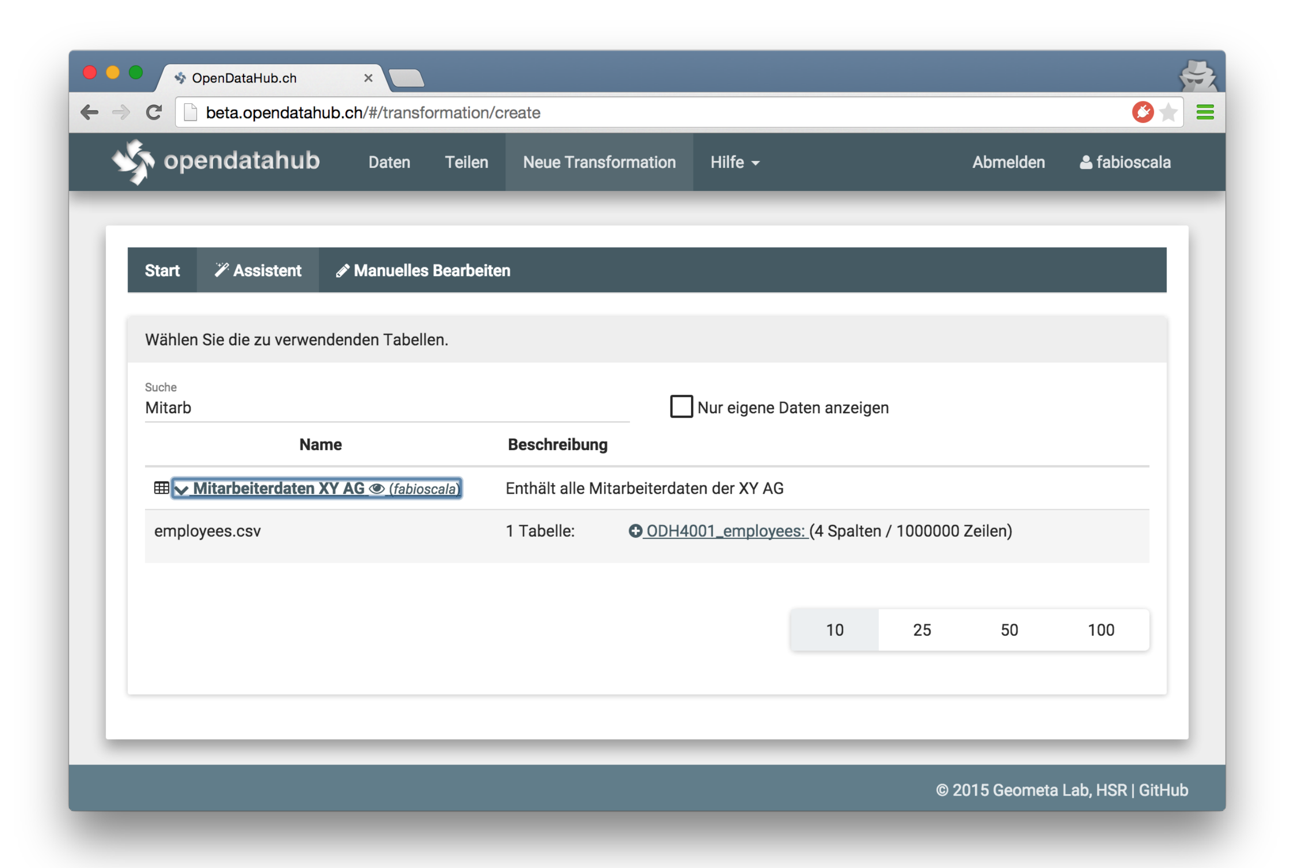


Abbildung 2.7: Daten zur Transformation selektieren

1. beta.opendatahub.ch → Neue Transformation → Assistent aufrufen

2. Mittels Suche die zu Transformierenden Daten in der Liste vorhandener Daten suchen.

3. Die Daten mittels Klick aufklappen und die gewünschte Tabelle wiederum durch Klick zur Transformation hinzufügen.

Feldselektion

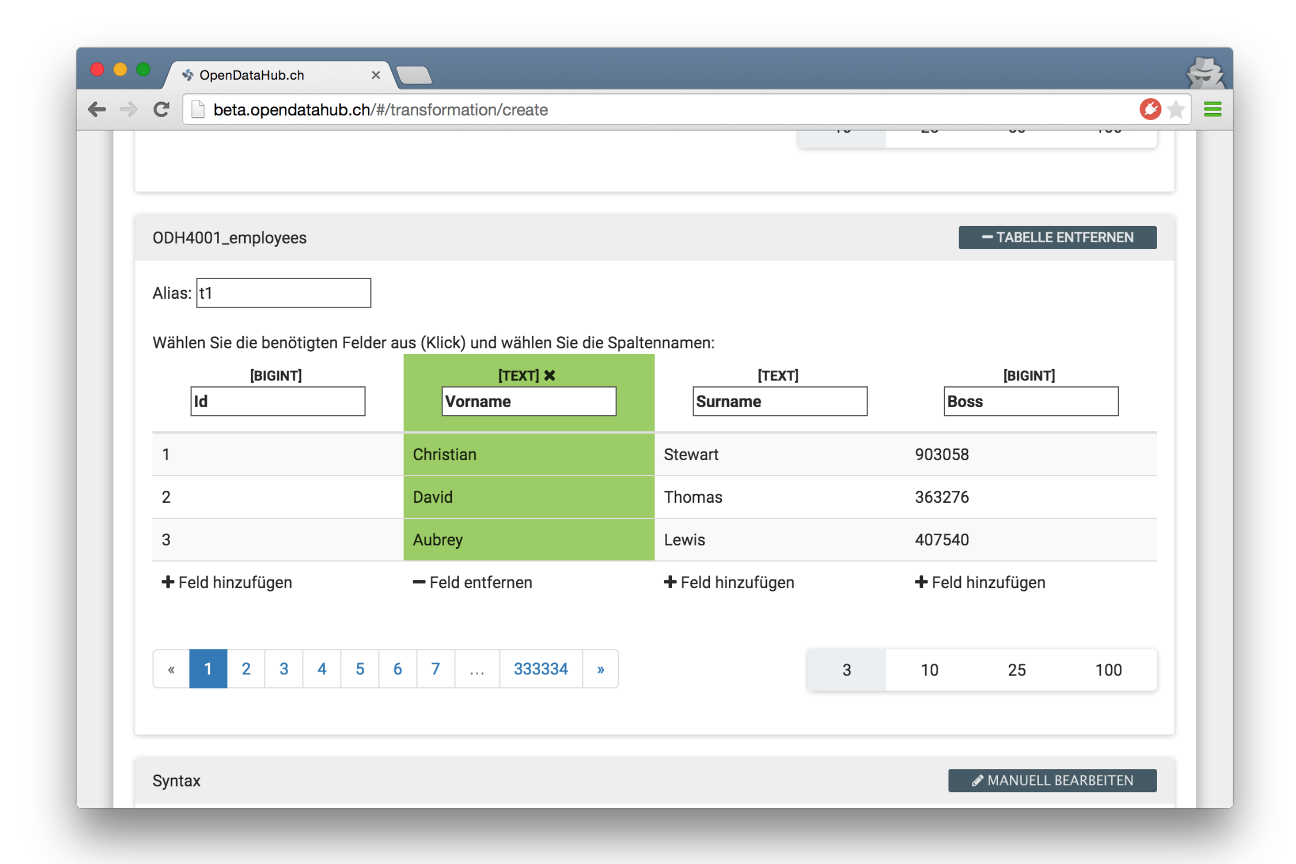


Abbildung 2.8: Selektion und Umbenennung einzelner Felder

1. Nachdem die Input-Daten im vorherigen Schritt hinzugefügt wurden, können nun einzelne Felder aus diesen Tabellen mittels Klick selektiert werden.

2. Auch der Name der Spalten kann direkt im Assistenten oberhalb der Spalten angepasst werden.

3. Als nächstes kann man wie in **??** abgebildet die zur Transformation generierte ODHQL-Abfrage betrachten sowie diese ausführen um eine Vorschau der Daten darzustellen.

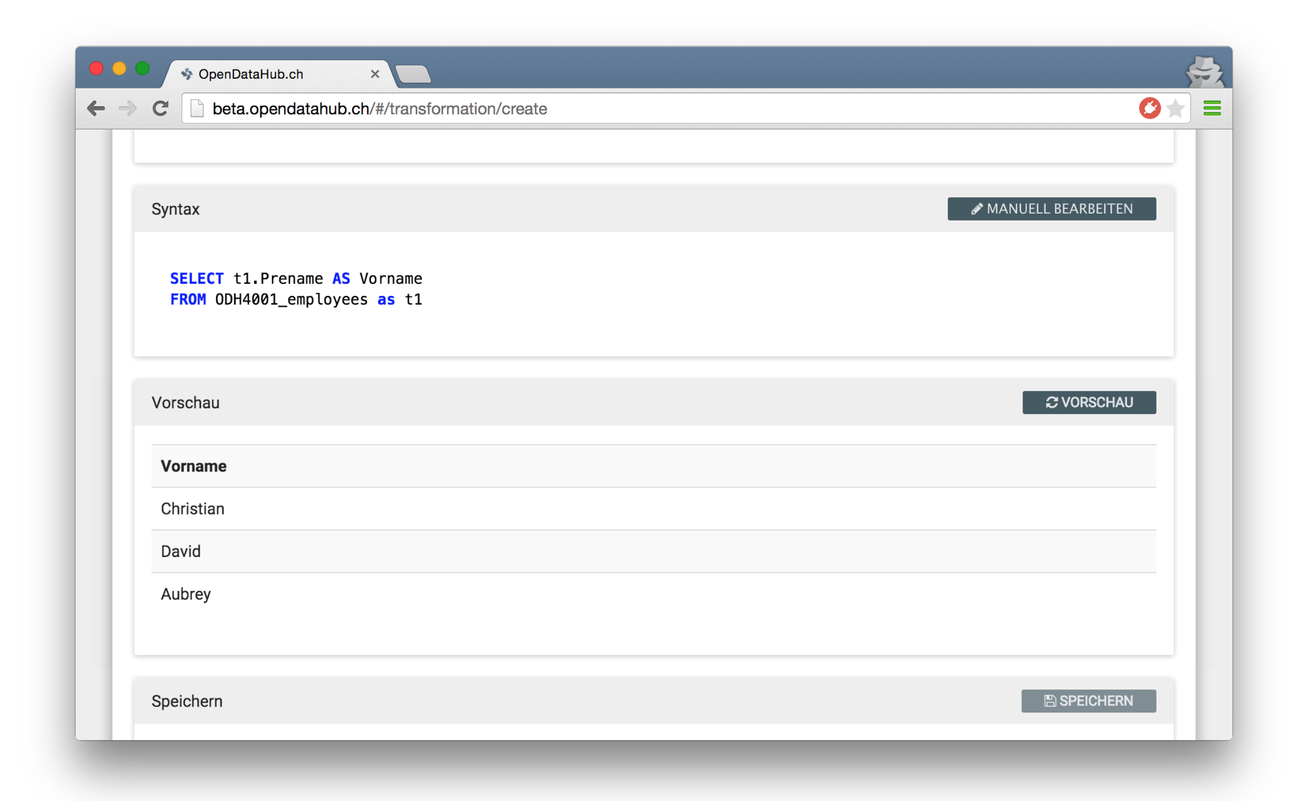


Abbildung 2.9: Vorschau der Transformation

Speichern

Analog wie beim Teilen von Daten (in **??** beschrieben), muss die Transformation vor der Freigabe noch mit zusätzlichen Attributen wie Name und Beschreibung versehen werden.

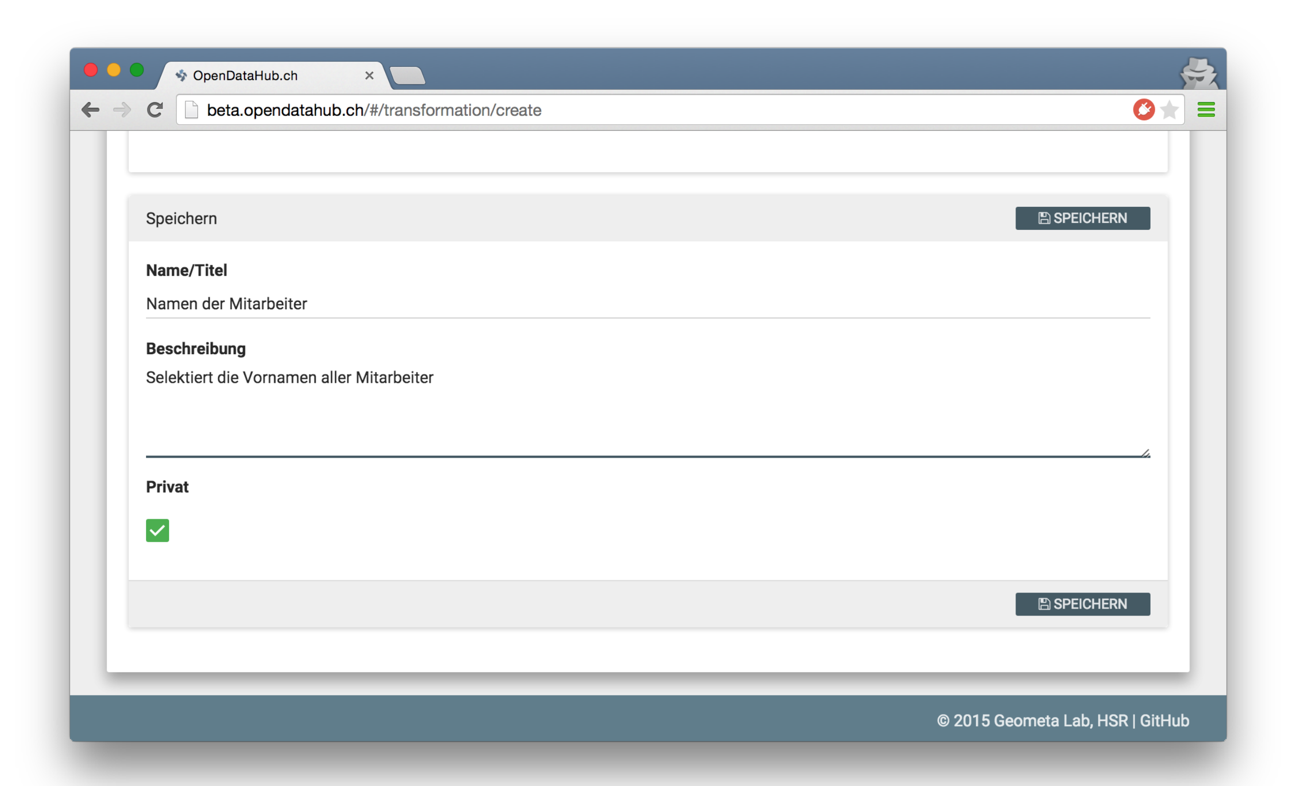


Abbildung 2.10: Speichern der Transformation

2.4.2 Manuelle Abfrage

- Nach der Verwendung des Assistenten kann die erstellte Abfrage manuell erweitert werden, um die volle Mächtigkeit von ODHQL-Funktionen nutzen zu können.

- Durch Klick auf "Manuelles Bearbeiten", in **??** ersichtlich, kann die Abfrage direkt bearbeitet werden.

- Durch Klick auf "Hilfe" oder beta.opendatahub.ch → Hilfe → ODHQL Referenz oder in **??** gelangt man auf die Dokumentation der Abfrage-Sprache.

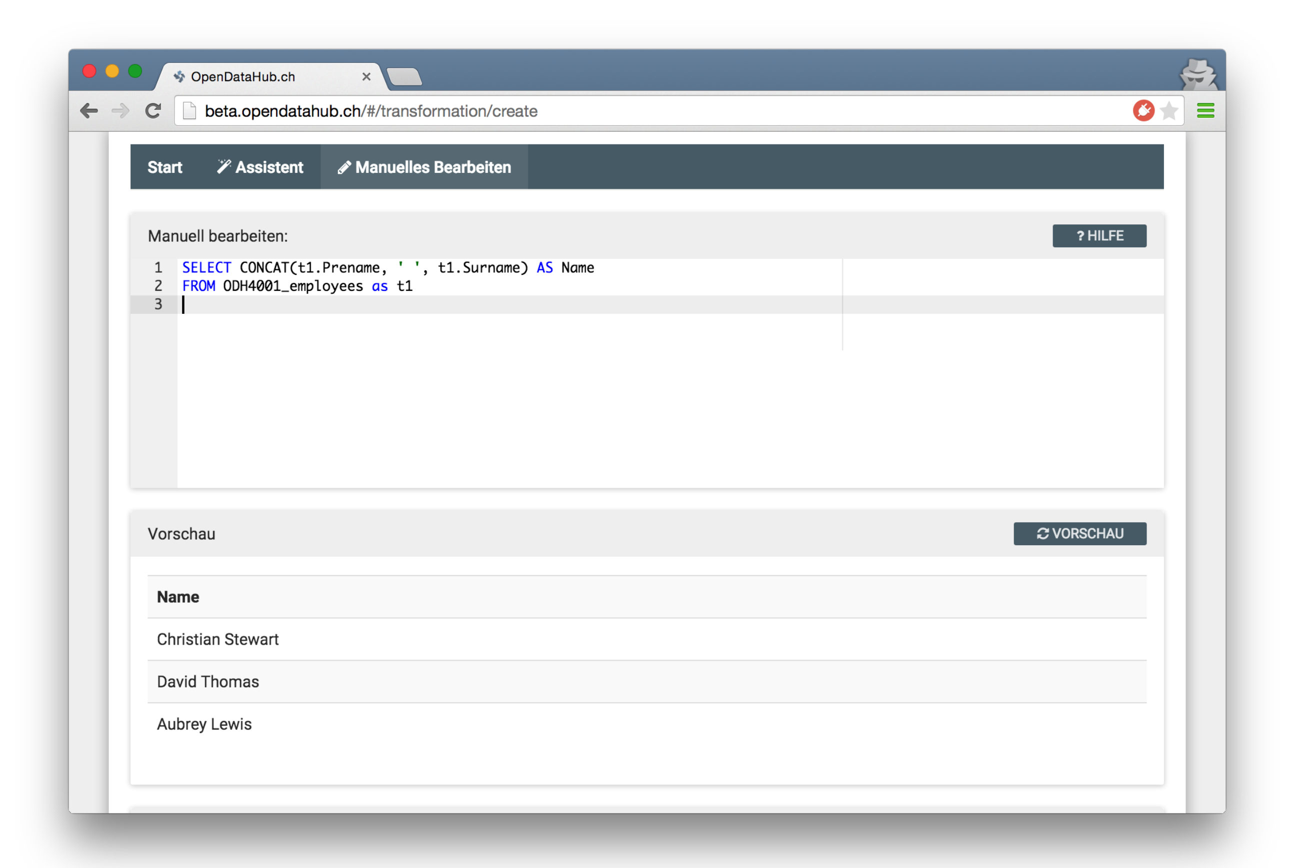


Abbildung 2.11: Manuelle ODHQL Abfrage

?? zeigt eine Erweiterung der mit dem Assistenten erstellten Abfrage, wobei Vor- und Nachname zusammengeführt werden um so die Spalte "Name" zu erzeugen.

A Inhalt der CD

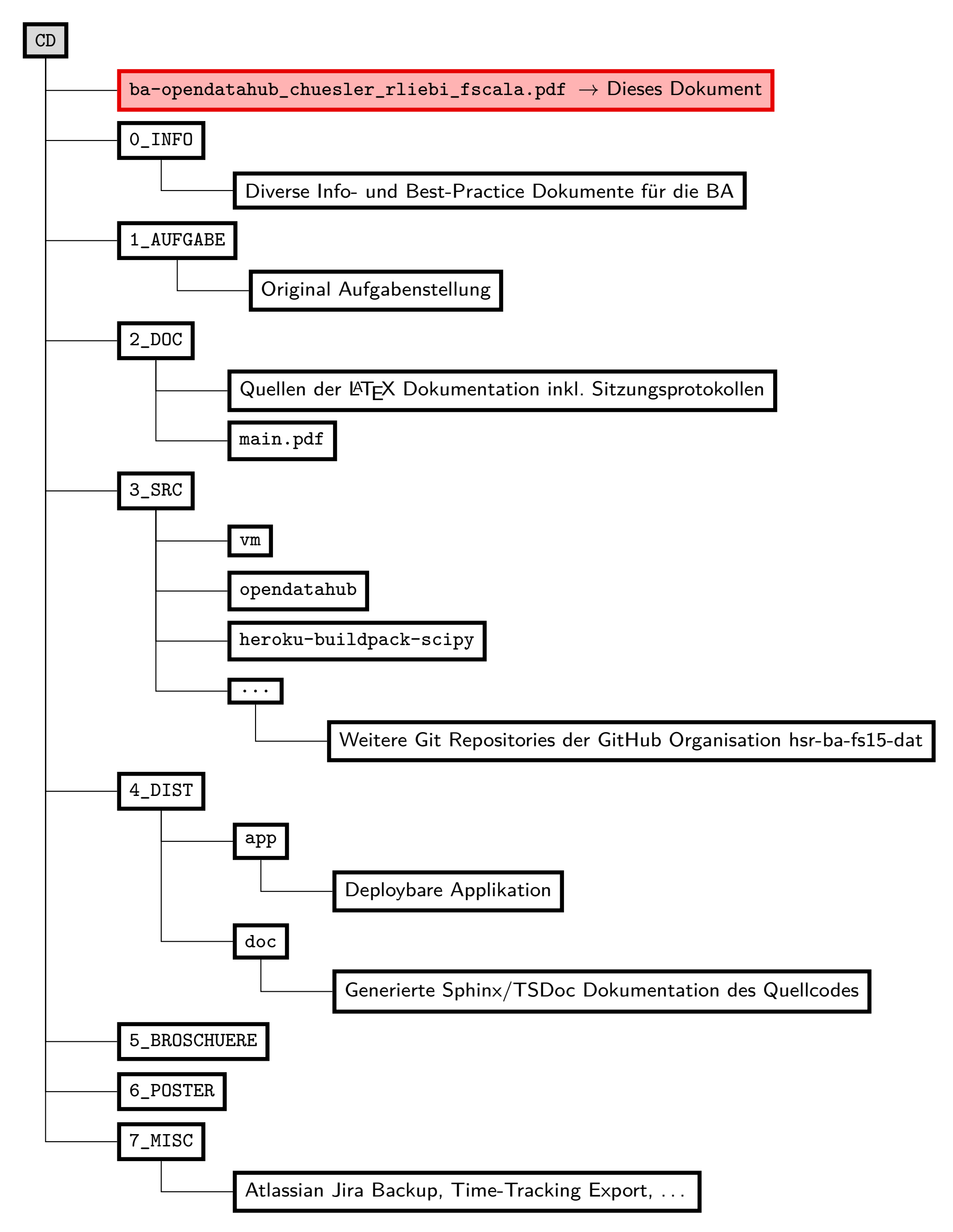


Abbildung A.1: Inhalte der beigefügten CD

B Eigenständigkeitserklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt sind oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Ort, Datum:

Rapperswil, 1. Juli 2015

Namen, Unterschriften:

Fabio Scala	Christoph Hüsler	Remo Liebi

C dat - Schnittstellen

Dieses Kapitel beschreibt die Möglichkeiten wie dat mittels Schnittstellen und Konfiguration über dessen Basisfunktionalität eines simplen tabellen-artigen Speichers hinaus verwendet werden kann.

C.1 CLI

```
$ dat help

Example usage

make a new folder and turn it into a dat store

    mkdir foo
    cd foo
    dat init

put a JSON object into dat

    echo '{"hello": "world"}' | dat import --json

stream the most recent of all rows

    dat cat

stream a CSV into dat

    cat some_csv.csv | dat import --csv

or

    dat import --csv some_csv.csv

use a custom newline delimiter:

    cat some_csv.csv | dat import --csv --newline $'\r\n'

use a custom value separator:

    cat some_tsv.tsv | dat import --csv --separator $'\t'

stream NDJSON into dat

You can pipeline Newline Delimited JSON (
```

```
NDJSON (http://ndjson.org/)
) into dat on stdin and it will be stored

    cat foo.ndjson | dat import --json

specify a primary key to use

    echo $'a,b,c\n1,2,3' | dat import --csv --primary=a
    echo $'{"foo":"bar"}' | dat import --json --primary=foo

attach a blob to a row

    dat blobs put jingles jingles-cat-photo-01.png

stream a blob from a row

    dat blobs get jingles jingles-cat-photo-01.png

add a row from a JSON file

    dat rows put burrito-recipe.json

get a single row by key

    dat rows get burrito

delete a single row by key

    dat rows delete burrito

but rows are never truly deleted. you can always go find a row at the version it was
 ↪  last seen. in this case, that row was at version 1

    dat rows get burrito 1

start a dat server

    dat listen

then you can poke around at the REST API:

    /api/changes
    /api/changes?data=true
    /api/metadata
    /api/rows/:docid
    POST /api/bulk content-type: application/json (newline separated json)

pull data from another dat

    dat pull http://localhost:6461

push data to another dat
```

```
    dat push http://localhost:6461

delete the dat folder (removes all data + history)

    dat clean

view raw data in the store

    npm install superlevel -g
    superlevel .dat/store.dat createReadStream
```

C.2 Datscript

Datscript ist eine DSL für dat die es ermöglicht Befehle zu konfigurieren sowie mehrere Befehle zu sogenannten Pipelines zu verknüpfen. Obschon Datscript an diversen Stellen erwähnt wird und ein Repository auf GitHub[1] besteht, so ist es bisher lediglich ein Konzept und wurde noch nicht implementiert. Es ist zu vermuten, dass Datscript gar nicht implementiert wird und durch das in **??** beschriebene Gasket ersetzt wurde.

Programmcode C.1: Datscript Beispiel

```
# validate cli args
args [url]

# import another datscript file
import foo.ds

# set env var for all run commands to use
env foo = "bar"

# run a pipeline (hoists)
run pipeline import

# run a command (from node_modules/.bin or PATH)
run foobar

# define main (default) pipeline
pipeline main
  run pipeline import

# define pipeline w/ piping
pipeline import
  run foobar | run dat import --json
```

[1] Lediglich ein Beispiel für das Konzept, keinerlei Programmcode vorhanden

C.3 Gasket

Gasket ist eine Node.js-basierte Pipeline-Engine. Die Konfiguration geschieht entweder im JSON-Format direkt in `package.json`, oder aber in einem Node.js-Modul. Unterstützt wird die Ausführung von externen Programmen, oder die Verwendung von Node.js-Modulen.
Ein Beispiel einer trivialen `package.json` Konfiguration wird in **??** aufgezeigt und gibt lediglich "HELLO WORLD" aus .

Programmcode C.2: Offizielles Gasket Beispiel

```
{
  "name": "my-test-app",
  "dependencies" : {
    "transform-uppercase": "^1.0.0"
  },
  "gasket": {
    "example": [
      {
        "command": "echo hello world",
        "type": "pipe"
      },
      {
        "command": "transform-uppercase",
        "type": "pipe"
      }
    ]
  }
}
```

C.4 Schnittstellen / API

C.4.1 REST

Der dat-Server kann von jeder dat-Instanz durch `$ dat listen` gestartet werden.
dat implementiert aktuell nur HTTP Basic Authentication mit einem einzelnen Benutzer, welcher in dat.json konfiguriert werden kann.
Falls ein Admin-Benutzer konfiguriert ist, können nur angemeldete Sessions Daten ändern, es können jedoch immer noch alle Daten von allen Benutzern gelesen werden.
SSL wird bisher nicht unterstützt.

GET /
Liefert die Webapplikation dat-editor aus.

Antwort GET /api
Liefert allgemeine Informationen zur dat-Instanz

Programmcode C.3: GET /api

```
{
    "dat":"Hello",
    "version":"6.9.6",
    "changes":2,
    "name":"dat-test",
    "rows":53,
    "approximateSize":{
        "rows":"3 kB"
    }
}
```

GET /api/rows

Liefert eine Liste von Einträgen (Default: 50 Einträge).

Programmcode C.4: Antwort GET /api/rows

```
{
    "rows": [
        {
            "prename":"Leonie", "surname":"Hahn",
            "street":"Via Camischolas sura", "nr":"86",
            "zip":"1135", "city":"Denens",
            "key":"ci6huzf52000057coxym5lgy7", "version":1
        },
        {
            "prename":"Manuela", "surname":"Eichel",
            "street":"Vallerstrasse", "nr":"115",
            "zip":"3765", "city":"Pfaffenried",
            "key":"ci6hv0d0a00005vco99jxr7w0", "version":1
        },
        ...
    ]
}
```

GET /api/rows/:key

Liefert den Eintrag mit dem angegebenen Schlüssel, oder eine Fehlermeldung.

Programmcode C.5: Antwort GET /api/rows/ci6huzf52000057coxym5lgy7

```
{
    "prename":"Leonie", "surname":"Hahn",
    "street":"Via Camischolas sura", "nr":"86",
    "zip":"1135", "city":"Denens",
    "key":"ci6huzf52000057coxym5lgy7", "version":1
}
```

POST /api/rows
Fügt einen neuen Datensatz hinzu. Die Daten müssen als JSON vorliegen.
Die Antwort besteht entweder aus einer Konflikt-Meldung, oder aus dem neu hinzu gefügten Datensatz.

Programmcode C.6: POST /api/rows

```
$ curl -H 'Content-Type: application/json' -d '{"prename":"Markus", "surname":"Maier",
 ↪  "street":"Dreibündenstrasse", "nr":"135", "zip":"6774", "city":"Dalpe"}' -X POST
 ↪  localhost:6461/api/rows
{"prename":"Markus", "surname":"Maier", "street":"Dreibündenstrasse", "nr":"135",
 ↪  "zip":"6774", "city":"Dalpe", "key":"ci6hw717u0004x0cocfquv0j9", "version":1}
```

GET /api/rows/:key/:filename
Liefert das BLOB mit dem entsprechenden Filenamen, oder eine Fehlermeldung falls nicht verfügbar.

POST /api/rows/:key/:filename
Fügt ein neues BLOB zum angegebenen Datensatz hinzu. Der Datensatz muss bereits existieren, und die aktuelle Version des Datensatzes muss im Query-String angegeben werden:
`POST /api/rows/foo/photo.jpg?version=1 HTTP/1.1`

GET /api/session
Liefert Informationen zur aktuellen Session. Dieser Aufruf kann auch zur Anmeldung per Basic Authentication verwendet werden.

GET /api/login
Selbe Funktionalität wie /api/session, setzt jedoch den HTTP-Header “Basic realm=SSecure Area“', so dass Browser ein Login-Fenster anzeigen.

GET /api/logout
Zerstört die aktuelle Session und entfernt Client-Side Cookies.

GET /api/changes
Liefert eine Json-formatierte Version des Change Streams (siehe auch JavaScript-API createChangesStream).

GET /api/csv
Liefert eine CSV-Datei mit der letzten Version der Daten.

POST /api/bulk
Ermöglicht das Einfügen von mehreren Datensätzen gleichzeitig. Unterstützt werden JSON (Content-Type `application/json`) und CSV (Content-Type `text/csv`).
Falls die Daten akzeptiert werden besteht die Antwort aus einer Liste von JSON-Objekten mit Key und Version für die neu eingefügten Datensätze, andernfalls aus einem HTTP-Fehler.

Programmcode C.7: GET /api/metadata

```
{
    "changes":15,
    "liveBackup":false,
    "columns":["prename","surname","street","nr","zip","city","version:"]
}
```

GET /api/manifest
Liefert ein JSON-Objekt mit Informationen zum DB-Backend. Dies wird für RPC verwendet (siehe `/api/rpc`).

POST /api/rpc
Ein Server-Endpunkt für multilevel (Node.js-Modul).

GET /api/metadata
Liefert Informationen zum Schema. Diese Daten werden während der Replikation verwendet.

GET /api/replicator/receive und GET /api/replicator/send
Wird für Replikation verwendet.

C.4.2 JavaScript

dat kommt als Node.js-Modul daher und ist die einzige Möglichkeit überhaupt um auf performante Art und Weise das dat Repository zu manipulieren. Das CLI verwendet intern ebenfalls die JavaScript API. Dieser Abschnitt zeigt auf welche Interaktion mit dem Repository via JavaScript API möglich ist.

Programmcode C.8: Laden des dat-Moduls

```
var dat = require('dat')
```

```
var db = dat([path], [options], [onReady])
```

Erstellt eine neue oder öffnet eine bereits bestehende dat-Datenbank. Alle Parameter sind optional.

path (string) Pfad zum Verzeichnis, welches das `.dat`-Verzeichnis enthält. Falls kein `.dat`-Verzeichnis existiert wird ein neues erstellt. Default ist `process.cwd()`.

options (object) Weitere Konfigurations-Parameter:

- **init** Wenn `false` erstellt dat keine neue Datenbank beim Initialisieren. Default: `true`.
- **storage** Wenn `false` versucht dat nicht beim Initialisieren die Datenbank zu lesen. Default: `true`.

path Alternative zum Konstruktor-Argument.

adminUser, adminPass Wird als Benutzer/Passwort verwendet für HTTP Basic Authentication wenn beide konfiguriert sind.

leveldown Benutzerdefiniertes `leveldown`-Backend. Default: `require('leveldown-prebuilt')`

db Benutzerdefinierte `levelup`-Instanz. Wenn diese Option vorhanden ist wird die `leveldown`-Option ignoriert und alle Tabellen-basierte Daten werden in dieser DB-Instanz gespeichert.

blobs Benutzerdefinierte `blob`-Datenbank. Default: `require('lib/blobs.js')`

replicator Benutzerdefinierte Replicator-Instanz. Default: `require('lib/replicator.js')`

remoteAddress Falls angegeben startet dat im RPC Client-Modus. Default: `undefined`

manifest Wenn `remoteAddress` gesetzt ist wird dies als Manifest für RPC verwendet.

skim Wenn `true` wird lazy auf BLOBs von einer nicht-lokalen Quelle zugegriffen. Default: `false`.

transformations siehe Transformationen

hooks siehe Transformationen

onReady: (err) Wird aufgerufen nachdem dat initialisiert wurde. Wenn `err` gesetzt ist trat ein Fehler auf.

Transformationen Transformationen können vor `put`- oder nach `get`-Operationen ausgeführt werden. Die Konfiguration erfolgt über den `options`-Parameter im Konstruktor.

Programmcode C.9: Beispiel einer Transformations-Konfiguration

```
{
  "transformations": {
    "get": "transform-uppercase",
    "put": [{"module": "./lowercase-stream.js"}]
  }
}
```

Folgende Datentypen sind möglich:

string Ausführbarer Befehl, um die Daten zu transformieren. Der Befehl erhält durch Zeilenumbrüche getrennte JSON-Datensätze (NDJSON) in `STDIN`. Nach der Transformation werden die Daten wieder als JSON auf `STDOUT` erwartet.

object Objekt mit einem der folgenden Feldern:

command Gleiche Funktionalität wie oben

module Per `require()` ladbares Node.js-Modul. Das Modul muss einen Streams2-Passthrough Stream exportieren mit `objectMode: true`. Falls das Modul installiert werden muss, sollte die Abhängigkeit direkt in package.json eingetragen werden.

array Array von Transformationen, welche nacheinander ausgeführt werden.

Hooks Aktuell existiert nur ein einzelner Hook:

listen Wird ausgeführt, wenn der dat-Server an einen Port bindet.

Ein Hook muss als Node.js-Modul, wie in **??** gezeigt, vorliegen.

Programmcode C.10: Hook-Beispiel

```
module.exports = function hook(dat, done) {
  // do stuff with dat

  // must call done when the hook is done initializing, even if you call it
  ↪  immediately
  done()
}
```

Nach erfolgreicher Initialisierung muss die Funktion `done` zwingend aufgerufen werden!

`get(key, [options], callback: (error, value))`
Findet den zum Key gehörenden Datensatz, falls vorhanden, und übergibt das Resultat an den callback.

options Objekt mit folgenden Feldern:

version Erlaubt das finden einer spezifischen Version. Default: aktuellste Version.

`put([key], value, [options], callback: (error, newVersion))`
Fügt einen neuen Datensatz in die Datenbank ein. Der Key kann optional als Parameter oder als `value.key` übergeben werden.
Bereits bestehende Einträge werden nur überschrieben, wenn `value.version` mit der aktuellsten in der Datenbank existierenden Version übereinstimmt. Andernfalls tritt ein Konflikt-Fehler auf.

options Objekt mit folgenden Feldern:

force Wenn `true` wird der Versions-Check übersprungen und Konflikte ignoriert. Für die neuen Daten wird eine neue Version erzeugt.

`delete(key, callback: (error, newVersion))`
Markiert den Key als gelöscht. Achtung: Alte Versionen bleiben erhalten und können weiterhin abgerufen werden.

`var readStream = db.createReadStream([options])`
Liefert einen lesbaren Stream der neusten Versionen aller Datensätze.
Die Einträge werden als `{key: key, value: value}` geliefert.

options Objekt mit folgenden Feldern:

start Start-Key. Default: Erster Key.

end End-Key. Default: Letzter Key.

limit Anzahl Datensätze, die geliefert werden sollen. Default: Unlimitiert.

`var valueStream = db.createValueStream([options])`
Liefert einen lesbaren Stream über die Werte der aktuellsten Versionen.
Standardmässig wird auf dem Stream ein Objekt pro Zeile ausgegeben.

options Objekt mit folgenden Feldern: `createValueStream` unterstützt die selben Optionen wie `createReadStream`, sowie zusätzlich folgende:

format Wenn diese Option auf `csv` oder `json` gesetzt wird, werden die Daten serialisiert anstatt als Objekte geliefert.

csv Identisch zu `format:'csv'`.

json Identisch zu `format:'json'`

`var keyStream = db.createKeyStream([options])`
Liefert einen Stream über die Schlüssel der Datensätze. Der Stream liefert ein Objekt pro Zeile, in der Form `key: key, version: number, deleted: boolean`.
Optionen sind identisch zu `createReadStream`.

`var changes = db.createChangesStream([options])`
Liefert einen Stream über das dat-Changelog. Die gelieferten Objekte haben die Form `change: changeId, key: key, version: number`.

options Objekt mit folgenden Feldern:

data Wenn `true` enthalten die gelieferten Objekte das Attribut `value`. Default: `false`.

since Change-Id, ab welcher Daten geliefert werden sollen. Default: 0.

tail Wenn `true` wird `since` auf die letzte Change-Id gesetzt, so dass nur neue Änderungen geliefert werden. Default: `false`.

limit Limitiert die Anzahl der zu liefernden Änderungen. Default: unlimitiert.

live Wenn `true` werden neue Änderungen geliefert sobald sie auftreten und der Stream endet nicht (bzw. muss manuell geschlossen werden).

`var writeStream = db.createWriteStream([options])`
Liefert einen schreibbaren Stream. Jede Schreib-Operation liefert Status-Informationen als Objekt zurück.

options Objekt mit folgenden Feldern:

format Teilt dem Stream mit, welches Format geschrieben werden soll. Erlaubte Werte: `'csv'`, `'json'`, `'protobuf'` oder `'objectMode'` (default).

csv, json, protobuf Equivalent zu `format:'csv'`, `format:'json'` bzw. `format:'protobuf'`

primary Spalte bzw. Array von Spalten, welche als Primary Key verwendet werden soll. Default: `key`.

hash Wenn `true` wird als Primary Key der Hex-formatierte MD5-Hash der Primary Key-Spalten verwendet.

primaryFormat Funktion, welche den Key formattiert bevor er eingefügt wird. Als Rückgabewert muss ein String geliefert werden. Akzeptiert `(val)`.

columns Liste von Spalten-Bezeichnungen, welche für CSV/MultiBuffer verwendet werden sollen.

headerRow Muss auf `false` gesetzt werden, wenn der CSV-Input keine Titel-Zeile enthält. In diesem Fall sollte auch `columns` gesetzt

separator Feld-Separator für CSV. Default: `','`

delimiter Record-Separator für CSV. Default: `'\n'`

`var versions = db.createVersionStream(key, [options])`
Liefert alle Versionen zum angegebenen Key.

options Objekt mit folgenden Feldern:

start Start-Version.

end End-Version

`var blobWriter = db.createBlobReadStream(key, filename, [options])`
Liefert einen lesbaren Stream mit Blob-Daten.

options Objekt mit folgenden Feldern:

version Version des Datensatzes. Default: Aktuellste Version.

`var blobWriter = db.createBlobWriteStream(filename, [row], [callback])`
Liefert einen schreibbaren Stream, welcher Blob-Daten annimmt.

filename Ein String oder ein Objekt mit einem `filename`-Attribut: `filename:'example.png'`.

row Objekt, welches den Datensatz identifiziert an den dieses Blob angehängt werden soll. Es gelten die selben Regeln wie bei `put()`. Falls nicht angegeben wird ein neuer Datensatz erstellt.

callback: (error, updated) Wird nach der Schreiboperation mit dem aktualisierten Datensatz aufgerufen.

`dat.listen([port], [callback: (error)])`
Startet den HTTP-Server.

port Zu verwendender Port. Default: 6461, bzw. der nächsthöhere freie Port.

`dat.clone(remote, [callback])`
Initialisiert ein neues dat (falls nicht bereits vorhanden) und erstellt einen lokalen Klon von `remote`. Kann schneller sein als `pull()`, falls der Server schnellere clone-Fähigkeiten hat (z.B. `liveBackup` von hyperleveldb).

`dat.push(remote, [callback: (error)])`
Synchronisiert das lokale dat mit dem `remote`-Server, indem die lokalen Änderungen über HTTP gepusht werden.

remote HTTP-Basis-Adresse des Servers (z.B. `http://localhost:6461`).

`dat.pull([remote], [callback])`
Synchronisiert das lokale dat mit dem angegebenen Server, indem die Änderungen vom Server übernommen werden.

remote HTTP-Basis-Adresse des Servers. Default: Adresse des Server, von dem diese dat-Instanz geklont wurde, falls vorhanden.

`dat.init(path, [callback])`
Erstellt eine neue dat-Datenbank in `path/.dat`. Diese Methode wird Standardmässig aufgerufen bei der Erstellung einer dat-Instanz.

`var paths = dat.paths(path)`
Liefert ein Objekt mit diversen relevanten Pfaden, mit `path` als Basis.

Programmcode C.11: Beispiel: Pfade mit `.` als Basis

```
{
    dir: '.',
    dat: '.dat',
    level: '.dat/store.dat',
    port: '.dat/PORT',
    blobs: '.dat/objects',
    package: 'dat.json'
}
```

`dat.exists(path, callback: (error, exists))`
Prüft ob eine dat-Datenbank am angegebenen Pfad existiert.

`dat.close(callback: (error))`
Beendet den HTTP-Server, den RPC-Client (falls vorhanden) und die Datenbank und räumt die `.dat/PORT`-Datei auf.

`dat.destroy(path, callback: (error))`
Ruft `close()` auf und entfernt das `.dat`-Verzeichnis in `path`.

`var headers = dat.headers()`
Liefert ein Array mit den aktuellen Spalten-Namen.

dat.getRowCount(callback: (error, count))
Ermittelt die aktuelle Anzahl Datensätze in der dat-DB.

C.4.3 Python

Mit datPython[2] existiert eine sehr simple Python API welche folgende grundlegenden Operationen mittels der REST Schnittstelle von dat ermöglicht.

info() Generelle Informationen über das dat Repository.

diff() Gibt alle Änderungen zurück.

csv() Gibt die Daten aus dem dat Repository als Comma Separated Values (CSV) formatierte Zeichenkette zurück.

rows() Gibt alle Zeilen im dat Repository zurück.

dict() Dasselbe wie `rows()`, jedoch als Dictionary.

put_json() Importiert alle Datensätze einer JSON Datei.

put_csv() Importiert alle Datensätze einer CSV Datei.

datPython ist zurzeit lediglich ein minimaler HTTP Wrapper und besteht aus knapp 60 Programmcodezeilen. Die Installation mittels Python Package Index (PyPI) bzw. `pip` wäre zwar möglich, jedoch nicht funktionsfähig. Bei Verwendung dieses noch kleinen Moduls wäre die Erstellung eines eigenen Forks zum jetzigen Zeitpunkt die beste Lösung.

Programmcode C.12: datPython Verwendung

```
>>> from datPython import Dat
>>> dat = Dat('http://7hhtoqpk6c8wu3di.c.try-dat.com')

>>> dat.info()
{"dat":"Hello","version":"6.8.4","changes":2,"name":"root","rows":1,
 ↪   "approximateSize":{"rows":"502 B"}}

>>> dat.diff()
{"change":1,"key":"schema","from":0,"to":1,"subset":"internal"}
{"change":2,"key":"ci6i1errr000012t5m9ggeh6h","from":0,"to":1}

>>> dat.csv()
key,version,name,age
ci6i1errr000012t5m9ggeh6h,1,alice,35

>>> dat.rows()
{"rows": [
    {"name":"alice","age":"35","key":"ci6i1errr000012t5m9ggeh6h","version":1}
    ]}
```

[2] https://github.com/pkafei/Dat-Python

```
>>> dat.dict()
{u'rows': [{u'age': u'35', u'version': 1, u'name': u'alice', u'key':
 ↪  u'ci6i1errr000012t5m9ggeh6h'}]}

>>> dat.put_json('test.json')
>>> dat.put_csv('test.csv')
```

D ODHQL-Syntax

⟨*UnionQuery*⟩ ::= ⟨*Query*⟩ (union ⟨*Query*⟩)* (⟨*OrderByList*⟩)?

⟨*Query*⟩ ::= ⟨*FieldSelectionList*⟩ ⟨*DataSourceSelectionList*⟩ (⟨*FilterList*⟩)?

⟨*FieldSelectionList*⟩ ::= select ⟨*FieldSelection*⟩ (, ⟨*FieldSelection*⟩)*

⟨*FieldSelection*⟩ ::= ⟨*Field*⟩ | ⟨*Expression*⟩ as Alias

⟨*CaseExpression*⟩ ::= case (when ⟨*Condition*⟩ then ⟨*Expression*⟩)+ (else ⟨*Expression*⟩)? end

⟨*Expression*⟩ ::= ⟨*Function*⟩ | ⟨*LiteralExpression*⟩ | ⟨*Field*⟩ | CaseExpression

⟨*Function*⟩ ::= ⟨*Identifier*⟩ ((⟨*FunctionArgumentList*⟩)?)

⟨*FunctionArgumentList*⟩ ::= ⟨*Expression*⟩ ((, ⟨*Expression*⟩)*)?

⟨*Field*⟩ ::= ⟨*DataSourceNameOrAlias*⟩ . FieldName

⟨*DataSourceNameOrAlias*⟩ ::= ⟨*DataSourceName*⟩ | Alias

⟨*DataSourceSelectionList*⟩ ::= from ⟨*DataSourceName*⟩ (as? ⟨*Alias*⟩)? (⟨*JoinDefinition*⟩)*

⟨*JoinDefinition*⟩ ::= (left | right | full)? join ⟨*DataSourceName*⟩ (as? ⟨*Alias*⟩)? on JoinCondition

⟨*JoinCondition*⟩ ::= ⟨*SingleJoinCondition*⟩ | (⟨*SingleJoinCondition*⟩ (and ⟨*SingleJoinCondition*⟩)*)

⟨*SingleJoinCondition*⟩ ::= ⟨*Expression*⟩ = Expression

⟨*FilterList*⟩ ::= where FilterAlternative

⟨*FilterAlternative*⟩ ::= ⟨*FilterCombination*⟩ (or ⟨*FilterCombination*⟩)*

⟨*FilterCombination*⟩ ::= ⟨*Condition*⟩ (and ⟨*Condition*⟩)*

⟨*Condition*⟩ ::= ⟨*BinaryCondition*⟩ | ⟨*InCondition*⟩ | ⟨*IsNullCondition*⟩ | ⟨*PredicateCondition*⟩
| (⟨*FilterAlternative*⟩)

⟨*BinaryCondition*⟩ ::= ⟨*Expression*⟩ ⟨*BinaryOperator*⟩ Expression

⟨*BinaryOperator*⟩ ::= = | != | <= | <| >= | > | (not)? like

⟨*InCondition*⟩ ::= ⟨*Expression*⟩ (not)? in (⟨*Expression*⟩ (, ⟨*Expression*⟩)*)

⟨*IsNullCondition*⟩ ::= ⟨*Field*⟩ `is` (`not`)? Null

⟨*PredicateCondition*⟩ ::= (`not`)? Function

⟨*OrderByList*⟩ ::= `order by` ⟨*OrderByField*⟩ (`,` ⟨*OrderByField*⟩)*

⟨*OrderByField*⟩ ::= (⟨*Field*⟩ | ⟨*Alias*⟩ | Position) (`asc` | `desc`)?

⟨*Integer*⟩ ::= (`0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`)+

⟨*LiteralExpression*⟩ ::= ⟨*SingleQuotedString*⟩ | ⟨*Number*⟩ | ⟨*Boolean*⟩ | Null

⟨*Number*⟩ ::= ⟨*Integer*⟩ | Float

⟨*Float*⟩ ::= ⟨*Integer*⟩ `.` Integer

⟨*Boolean*⟩ ::= `true` | `false`

⟨*Null*⟩ ::= `null`

⟨*SingleQuotedString*⟩ ::= <string> <in> <single> quotes

⟨*DoubleQuotedString*⟩ ::= <string> <in> <double> quotes

⟨*DataSourceName*⟩ ::= Identifier

⟨*FieldName*⟩ ::= Identifier

⟨*Alias*⟩ ::= Identifier

⟨*Identifier*⟩ ::= (`a..z` | `A..Z` | `_`) (`a..z` | `A..Z` | `_` | ⟨*Integer*⟩)* | DoubleQuotedString

OpenDataHub Query Language (ODHQL)

Inhalt

- UPPER(strings)
- XPATH(values, path)

Was ist ODHQL?

ODHQL ist eine an SQL angelehnte Abfrage- und Transformations-Sprache für OpenDataHub.

Syntax

Bestandteile einer Abfrage

Eine Abfrage besteht aus folgenden Teilen:

- Eine Liste von Feldern oder Ausdrücken, welche im Resultat erscheinen sollen
- Eine Liste von Datenquellen
- Optional eine Liste von Filter-Ausdrücken
- Optional eine Sortier-Klausel

Gross- und Kleinschreibung wird nicht beachtet.

Mehrere Abfragen können kombiniert werden mithilfe von Union. In diesem Fall ist nur eine Sortier-Klausel am Ende der kombinierten Abfrage erlaubt.

```
SELECT NULL AS userid,
           -- Null-Ausdruck
       SUBSTRING(NVL(EXTRACT(t.text, '\|([^|\.]+)'), 'no value'), 1, 100)
AS title,  -- Funktions-Aufruf
       EXTRACT(t.text, '\|([^|]+)') AS description,
       CAST(CAST(t."df", 'bigint'), 'datetime') AS trob_start,
       CAST(CAST(t."dd", 'bigint'), 'datetime') AS trob_end,
       NULL AS trob_interval,
       'both' AS direction,
           -- String-Ausdruck
       NULL AS diversion_advice,
       CASE WHEN t.p = 'Switzerland' THEN 'CH'
           -- Case-Ausdruck
            WHEN t.p = 'France' THEN 'FR'
            WHEN t.p = 'Austria' THEN 'AT'
       END AS country,
       t.text AS reason,
           -- Feld
       EXTRACT(t.text, '\|([^|,]+)') AS object_name,
       'street' AS object_type,
       CASE WHEN CONTAINS(t.text, 'closed', FALSE) THEN 'closed'
            WHEN (CONTAINS(t.text, 'maintenance', FALSE) OR CONTAINS(t.tex
t, 'maintenance', FALSE))
                THEN 'obstructed'
            ELSE 'other'
       END AS trob_type,
       ST_SETSRID(ST_GEOMFROMTEXT(CONCAT('POINT(', t.x, ' ', t.y, ')')), 3
395) AS trobdb_point
```

```
  FROM ODH11 AS t
           -- Datenquelle
 ORDER BY 4
```

Felder und Ausdrücke

In der Feld-Liste sind folgende Ausdrücke erlaubt:

Feld

Bezieht sich direkt auf ein Feld einer Datenquelle. Der Name des Feldes muss mit dem Namen oder Alias der Datenquelle prefixed werden. Optional kann ein Alias angegeben werden.

```
prefix.feld AS alias
```

Wert

Ganzzahl (Integer), Fliesskommazahl (Float, Trennzeichen ist ein Punkt), Zeichenkette (String, in einfachen Anführungszeichen), Boolean (true, false) oder Null. Es muss ein Alias angegeben werden.

Einfache Anführungszeichen können innerhalb eines Strings mit "\'" erzeugt werden, Backslashes ("\") mit "\\".

Funktion

Besteht aus einem Namen und einer Liste von Argumenten. Es muss zwingend ein Alias angegeben werden.

```
SUBSTRING(NVL(EXTRACT(t.text, '\|([^|\.]+)'), 'no value'), 1, 100) AS title
```

Fallunterscheidung (Case-Ausdruck)

Kann verwendet werden, um Werte zu übersetzen. Es muss mindestens eine Bedingung angegeben werden. Das Format ist 'when <Bedingung> then <Ausdruck>', wobei alle unten beschriebenen Bedingungs-Arten sowie hier beschriebenen Ausdrücke erlaubt sind.

```
CASE WHEN CONTAINS(t.text, 'closed', FALSE) THEN 'closed'
     WHEN (CONTAINS(t.text, 'maintenance', FALSE) OR CONTAINS(t.text, 'maintenance', FALSE))
        THEN 'obstructed'
     ELSE 'other'
END
```

Datenquellen

Es muss mindestens eine Datenquelle angegeben werden. Falls mehrere Datenquellen verwendet werden sollen, muss eine Verknüpfungsbedingung angegeben werden.

Unterstützt werden folgende Join-Typen:

Inner

Standard. Verlangt, dass beide Seiten vorhanden sind

```
FROM ODH12 AS employees
JOIN ODH13 AS employers ON employees.employer_id = employers.id
```

Left

Verwendet die Schlüssel der linken Seite für den Vergleich (rechte Seite kann null sein)

```
FROM ODH12 AS employees
LEFT JOIN ODH13 AS employers ON employees.employer_id = employers.id
```

Right
Verwendet die Schlüssel der rechten Seite für den Vergleich (linke Seite kann null sein)

```
FROM ODH12 AS employees
RIGHT JOIN ODH13 AS employers ON employees.employer_id = employers.id
```

FULL
Verwendet die Vereinigungsmenge der Schlüssel der beiden Seiten für den Vergleich (beide Seiten sind optional, es kann jedoch pro Zeile nur eine Seite null sein).

```
FROM ODH12 AS employees
FULL JOIN ODH13 AS employers ON employees.employer_id = employers.id
```

Filter

Folgende Filter-Ausdrücke sind möglich:
is null*, *is not null
Prüft ob ein Feld (nicht) null ist

in*, *not in
Prüft ob ein Ausdruck (nicht) in einer Liste enthalten ist. .. code:: sql

> country IN ('CH', 'DE', 'AT')

<, >, <=, >=, =, !=
Vergleicht zwei Ausdrücke miteinander.

like*, *not like
Prüft ob ein Ausdruck einem Muster entspricht. Verwendet wird dazu ein Regulärer Ausdruck mit Python Syntax.

Prädikat
Eine Funktion, welche ein boolsches Resultat liefert kann direkt als Bedingung verwendet werden.

Mehrere Bedingungen können mit 'and' und 'or' verknüpft und mit runden Klammern gruppiert werden.

```
WHERE t.a IS NOT NULL
  AND (t.b IN (1, 2, 3) OR t.b > 20)
```

Sortier-Klausel

Es kann sortiert werden nach Feld-Name, Alias oder Position in der Feld-Liste.

```
ORDER BY 1, ODH4.name DESC, surname ASC
```

Union

Die Resultate mehrerer Abfragen können mithilfe von Union kombiniert werden. Zu beachten sind folgende Punkte: - Union verhält sich wie UNION ALL in SQL, d.h. es wird keine Deduplizierung der Einträge vorgenommen - Die einzelnen Abfragen müssen kompatible Feldlisten liefern, d.h. gleiche Feld-Zahl und Feld-Typen. - Sollten Geometrie-Spalten vorhanden sein, so werden die Geometrien in das Referenzsystem der ersten Abfrage umgewandelt. Es ist somit nötig, dass für alle Geometrien ein Referenzsystem bekannt ist.

Als Feld-Namen werden im Resultat die Feld-Namen der ersten Abfrage verwendet.

Datentypen

ODHQL unterstützt zurzeit folgende Datentypen:

> **INTEGER**
> 32-Bit Ganzzahl-Wert
>
> **BIGINT**
> 64-Bit Ganzzahl-Wert
>
> **SMALLINT**
> 16-Bit Ganzzahl-Wert
>
> **FLOAT**
> 64-Bit Fliesskommazahl
>
> **DATETIME**
> Datum und Zeit
>
> **TEXT**
> Zeichenkette beliebiger Länge
>
> **GEOMETRY**
> Geometrie-Objekte. Nicht näher spezifiziert, d.h. es können Geometrien verschiedener Art in einer Spalte vorhanden sein. Der Experte ist selbst dafür Verantwortlich, dass gleiche Typen bei einer Transformation herauskommen, sodass diese für alle Zielformate kompatibel ist.

Hierbei ist wichtig anzumerken, dass diese Typen keine Keywords der Sprache sind, sondern beispielsweise bei der Verwendung der CAST() Funktion als Zeichenkette (z.B. 'INTEGER') übergeben werden müssen.

Funktionen

CAST(values, datatype)

Konvertiert den Datentyp einer Spalte oder eines einzelnen Wertes.

Parameter

- *values*: Spalte oder Wert der konvertiert werden soll.
- *datatype*: Gültiger ODHQL Datentyp

Beispiel

```
CAST(ODH42.age, 'INTEGER') AS age
```

CONCAT(a, b, ...args)

Konkateniert eine Liste von TEXT-Spalten oder Werten.

Parameter

- *args*: Liste von TEXT-Spalten oder -Werten

Beispiel

```
CONCAT(ODH5.given_name, ' ', ODH5.surname) AS name
```

CONTAINS(strings, pattern, match_case=True)

Prüft ob eine Zeichenkette in einem Text enthalten ist.

Kann als Bedingung verwendet werden.

Parameter

- *strings*: Spalte oder Wert in welchem gesucht werden soll
- *pattern*: Spalte oder Wert welcher gesucht werden soll
- *match_case*: Optional. Gibt an ob Gross- und Kleinschreibung beachtet werden soll. Default: True.

Beispiel

```
CONTAINS(ODH9.haltestelle, 'Hauptbahnhof') AS ist_hb
```

COUNT(strings, pattern)

Zählt die Anzahl Vorkommnisse des Musters im Text.

Parameter

- *strings*: Spalte oder Wert
- *pattern*: Regulärer Ausdruck. Siehe Regex-Dokumentation.

Beispiel

```
COUNT(ODH30.description, '\d') AS numbers
```

ENDSWITH(strings, end)

Prüft ob ein Text mit einer angegebenen Zeichenkette endet.

Kann als Bedingung verwendet werden.

Parameter

- *strings*: TEXT-Spalte oder -Wert
- *end*: TEXT-Spalte oder -Wert

Beispiel

```
ENDSWITH(ODH9.haltestelle, 'Flughafen') AS ist_flughafen
```

EXTRACT(strings, pattern, group=1)

Wendet einen Regulären Ausdruck (Regex) auf die angegebene Spalte oder den Wert an und liefert das Resultat der angegebenen Gruppe zurück.

Da Backslashes ('')

Parameter

- *strings*: TEXT-Spalte oder -Wert
- *pattern*: Regulärer Ausdruck. Siehe Regex-Dokumentation.
- *group*: Optional. Gruppe, welche ausgewertet werden soll. Default: 1.

Beispiel

```
EXTRACT(t.text, '\|([^|\.]+)') AS title
```

GET(strings, index)

Liefert das Zeichen an der angegebenen Stelle im Text (0-basiert).

Parameter

- *strings*: Spalte oder -Wert
- *index*: Spalte oder Wert.

Beispiel

```
GET(ODH12.country, 1) AS c
```

LEN(strings)

Liefert die Länge des TEXTs.

Parameter

- *strings*: TEXT-Spalte oder -Wert

Beispiel

```
LEN(ODH7.description) AS description_length
```

LOWER(strings)

Konvertiert alle Buchstaben in Kleinbuchstaben.

Parameter

- *strings*: TEXT-Spalte oder -Wert

Beispiel

```
LOWER(ODH7.ort) AS ort
```

LTRIM(strings)

Entfernt White Space vom Beginn der Spalte oder des Wertes.

Parameter

- *strings*: TEXT-Spalte oder -Wert

Beispiel

```
LTRIM(ODH7.strasse) AS strasse
```

NVL(a, b)

Gibt das zweite Argument zurück, falls das erste NULL ist.

Parameter

- *a*: Spalte oder Wert der auf NULL geprüft werden soll
- *b*: Spalte oder Wert der als Ersatz verwendet werden soll

Beispiel

```
NVL(ODH12.title, '') as title
```

PAD(strings, width, side)

Füllt die Zeichenkette auf die angegebene Länge mit Leerzeichen auf.

Parameter

- *strings*: Spalte oder Wert
- *width*: Gewünschte Länge
- *side*: Seite. Kann 'left', 'right' oder 'both' sein.

Beispiel

```
PAD(ODH4.name, 20, 'right') AS name
```

RANGE(start=1, step=1)

Liefert eine Sequenz von Ganzzahlen. Geeignet um beispielsweise künstliche IDs zu erstellen.

Parameter

- *start*: Erster Wert der Sequenz.
- *step*: Abstand zwischen den Ganzzahlen.

Beispiel

```
RANGE() AS id
```

REPEAT(strings, times)

Wiederholt die Zeichenkette eine bestimme Anzahl mal.

Parameter

- *strings*: Spalte oder Wert welcher wiederholt werden solle
- *times*: Anzahl Wiederholungen

Beispiel

```
REPEAT('Lorem ipsum dolor... ', 20) AS filler
```

REPLACE(strings, pattern, replace, match_case=True)

Ersetzt die angegebene Zeichenkette im Text.

Parameter

- *strings*: Spalte oder -Wert in welchem gesucht werden soll
- *pattern*: Spalte oder -Wert welcher gesucht werden soll
- *replace*: Ersatz-Spalte oder -Wert
- *match_case*: Optional. Gibt an ob Gross- und Kleinschreibung beachtet werden soll. Default: True.

Beispiel

```
REPLACE(ODH12.strasse, 'str.', 'strasse') AS strasse
```

ROUND(col, decimals)

Rundet auf die angegebene Anzahl Nachkommastellen.

Parameter

- *col*: Spalte oder Wert der gerundet werden soll. Muss vom Datentyp FLOAT sein.
- *decimals*: Anzahl Nachkommastellen, auf die gerundet werden soll.

Beispiel

```
ROUND(ODH20.fraction, 4) AS fraction
```

RTRIM(strings)

Entfernt White Space vom Ende der Spalte oder des Wertes.

Parameter

- *strings*: TEXT-Spalte oder -Wert

Beispiel

```
RTRIM(ODH7.strasse) AS strasse
```

ST_Area(geoms)

Berechnet die Fläche der Geometrien in der Einheit des Koordinatensystems.

Parameter

- *geoms*: Spalte mit Geometrien.

Siehe auch: PostGIS ST_Area

Beispiel

```
ST_X(ODH12.geometry) AS area
```

ST_AsText(geoms)

Liefert die Geometrien als Well-Known-Text (WKT) Zeichenkette,

Parameter

- *geoms*: Spalte mit Geometrien.

Siehe auch: PostGIS ST_AsText

Beispiel

```
ST_AsText(ODH12.geometry) AS wkt
```

ST_Centroid(geoms)

Berechnet den Mittelpunkt der Geometrien.

Parameter

- *geoms*: Spalte mit Geometrien.

Siehe auch: PostGIS ST_Centroid

Beispiel

```
ST_Centroid(ODH12.geometry) AS point
```

ST_GeomFromText(wkts, srid=None)

Erstellt eine Spalte mit Geometrie-Objekten aus einem Well-Known-Text (WKT).

Parameter

- *wkts*: Spalte oder Wert mit WKTs.
- *srid*: Optional die SRID der Geometrien.

Siehe auch: PostGIS ST_GeomFromText

Beispiel

```
ST_GeomFromText('POINT(7.2234283 48.8183157)', 4326) AS hsr
```

ST_SetSRID(geoms, srid)

Setzt das Koordinatensystem (Spatial Reference Identifier, kurz SRID) einer Geometrie-Spalte.

Parameter

- *geoms*: Spalte mit Geometrien.
- *srid*: SRID der Geometrien.

Siehe auch: PostGIS ST_SetSRID

Beispiel

```
ST_SetSRID(ODH12.latlng, 4326) AS geometry
```

ST_SRID(geoms)

Liefert das Koordinatensystem einer Geometrie-Spalte. Sollte keine Identifikationsnummer (SRID) bekannt sein, so wird eine Zeichenkette mit den Projektionsparametern im PROJ.4 <http://trac.osgeo.org/proj/>`_ Format zurückgegeben.

Parameter

- *geoms*: Spalte mit Geometrien.

Siehe auch: PostGIS ST_SRID

Beispiel

```
ST_SRID(ODH12.latlng) AS srid
```

ST_Transform(geoms, srid)

Transformiert die Geometrien in ein anderes Koordinatensystem. Auf den Geometrien muss bereits eine SRID gesetzt sein.

Parameter

- *geoms*: Spalte mit Geometrien.
- *srid*: SRID der Geometrien.

Siehe auch: PostGIS ST_Transform

Beispiel

```
ST_Transform(ODH12.mercator, 4326) AS latlng
```

ST_X(geoms)

Liefert die kleinste X-Komponente der Geometrien. **Achtung:** Funktioniert aus diesem Grund anders wie in

PostGIS nicht nur mit Punkten.

Parameter

- *geoms*: Spalte mit Geometrien.

Siehe auch: PostGIS ST_X

Beispiel

```
ST_X(ODH12.latlng) AS x
```

ST_Y(geoms)

Liefert die kleinste Y-Komponente der Geometrien. **Achtung:** Funktioniert aus diesem Grund anders wie in PostGIS nicht nur mit Punkten.

Parameter

- *geoms*: Spalte mit Geometrien.

Siehe auch: PostGIS ST_Y

Beispiel

```
ST_Y(ODH12.latlng) AS y
```

STARTSWITH(strings, start)

Prüft ob ein Text mit einer angegebenen Zeichenkette anfängt.

Kann als Bedingung verwendet werden.

Parameter

- *strings*: TEXT-Spalte oder -Wert
- *start*: TEXT-Spalte oder -Wert

Beispiel

```
STARTSWITH(ODH7.ort, 'Zürich') AS in_zurich
```

SUBSTRING(strings, start, length=None)

Liefert den angegebenen Teil des Texts.

Parameter

- *strings*: Spalte oder Wert
- *start*: Startposition (startet bei 1)
- *length*: Gewünschte Länge. Ohne Längenangabe wird der Rest des Texts zurückgegeben.

Beispiel

```
SUBSTRING(ODH30.description, 1, 100) AS title
```

TO_CHAR(values, format=None)

Konvertiert eine Spalte oder einen Werte zu TEXT. Für DATETIME-Spalten/Werte kann ein Format angegeben werden.

Parameter

- *values*: Spalte oder Wert
- *format*: Falls *values* vom Typ DATETIME ist. Siehe Dokumentation.

Beispiel

```
TO_CHAR(TO_DATE(ODH30.baubeginn, '%d%m%Y'), '%Y-%m-%d') AS baubeginn
```

TO_DATE(values, format=None)

Konvertiert ein Datum in TEXT-Form zu DATETIME.

Parameter

- *values*: Spalte oder Wert der konvertiert werden soll.
- *format*: Format-Angabe. Siehe Dokumentation.

Beispiel

```
TO_DATE(ODH5.baubeginn, '%d%m%Y') AS baubeginn
```

TRIM(strings)

Entfernt White Space vom Beginn und Ende der Spalte oder des Wertes.

Parameter

- *strings*: TEXT-Spalte oder -Wert

Beispiel

```
TRIM(ODH7.strasse) AS strasse
```

UPPER(strings)

Konvertiert alle Buchstaben in Grossbuchstaben.

Parameter

- *strings*: TEXT-Spalte oder -Wert

Beispiel

```
UPPER(ODH7.ort) AS ort
```

XPATH(values, path)

Wendet den XPath-Ausdruck auf die Spalte oder den Wert an.

Parameter

- *values*: Spalte oder Wert mit gültigem XML in TEXT-Form.
- *path*: XPath-Ausdruck. Zu beachten: Der Ausdruck darf genau ein Resultat pro Zeile in *values* liefern.

Beispiel

```
XPATH(t.description, '//tr[1]/td[2]/text()') AS abschnitt
```

F Sitzungsprotokolle

Die Sitzungsprotokolle wurden in dieser Ausgabe der Arbeit ausgelassen und sind nur in der vollen, auf der CD ablegten Version, einzusehen.